Intel® Pentium® II Processor Specification Update

Release Date: December 2000

Order Number: 243337-040

The Pentium® II processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
May 1997	-001	This document is the first Specification Update for the Pentium® II processor.
June 1997	-002	Added Erratum 25. Update Erratum 13 status in the Summary Table of Changes. Added Documentation Change Table and Documentation Change 1. Added 300-MHz Pentium II processor information.
July 1997	-003	Added Erratum 26. Added Specification Change Table and Specification Changes 1 and 2.
August 1997	-004	Added Erratum 27. Added Document Change 2 and Spec Changes 3, 4, 5, 6, and 7.
September 1997	-005	Updated Erratum 27. Added Errata 28 and 29. Added Document Change 3 and Spec Clarification 1. Added C1 stepping information. Updated Spec Change 6.
October 1997	-006	Updated Errata 6 and 18, and S-spec table.
November 1997	-007	Updated Erratum 22. Added Specification Clarification 2, 3, and 4.
December 1997	-008	Updated and added notes to S-spec table. Updated package information table. Updated Errata 24. Added Errata 30, 31, and 32.
January 1998	-009	Added notes to Pentium II processor markings. Updated Erratum 28. Added Erratum 33. Added Documentation Change 4 and 5. Added Specification Change 5.
January 26, 1998 (Special Edition)	-010	Updated S-spec table. Added dA0 stepping information. Added Errata 34, 35, 36, 37, and 38.
February 1998	-011	Added new processor markings. Corrected Errata 13 and 34 for steppings affected. Corrected typos in summary table for Errata 34, 35, and 36. Added Erratum 39. Added Documentation Change 6.
March 1998	-012	Added new boxed processor markings. Updated Documentation Changes section, Specification Clarifications section, and Specification Changes section. Corrected Erratum 8. Added Errata 40, and 41. Added Documentation Changes 6 and 7. Added Specification Clarification 6. Added Specification Changes 1 and 2.
April 1998	-013	Added new Mobile Pentium® II processor markings and Pentium II Mobile Modules markings. Updated Documentation Changes section, Specification Clarifications section, and Specification Changes section. Updated S-spec table. Added new steppings to Summary Table of Changes. Corrected Erratum 1. Added Errata 42, 43 and 44. Added Documentation Change 8. Updated Specification Change 1. Added Specification Change 3.
May 1998	-014	Updated S-spec table. Updated Errata 2 and 42. Added Errata 45 through 51. Corrected Documentation Change 7. Updated Specification Change 2.

REVISION HISTORY

Date of Revision	Version	Description
June 1998	-015	Updated S-spec Table. Updated Summary Table of Changes. Updated Erratum 47. Added Errata 52 and 53. Added Documentation Changes 9 through 16. Added Specification Clarifications 7 though 9. Updated Specification Change 1. Added Specification Change 4 and 5.
July 1998	-016	Added Pentium II Processor and Boxed Pentium II Processor 3 Line Markings. Updated Preface, Documentation Changes section, Specification Clarifications section, and Specification Changes section. Updated S-spec Table. Updated Summary Table of Changes. Added Errata 54 and 55. Added Documentation Changes 17 through 21. Added Specification Clarifications 10 through 15. Added Specification Change 6.
August 1998	-017	Moved all references to the Mobile Pentium II processor to the *Mobile Pentium® II Processor Specification Update*. Updated S-spec Table. Updated Summary Table of Changes. Updated Errata 6 and 38. Added Errata 56 through 59. Updated Specification Clarification 5.
September 1998	-018	Added new Pentium II OverDrive® processor markings. Updated S-spec table. Updated Errata 56 and 57. Added Errata 60 through 62. Added Specification Changes 6 and 7.
October 1998	-019	Implemented new numbering nomenclature. Updated S-spec table. Updated Errata A1 and A48. Added Errata A62, A63 and A64. Added Specification Change A8. Added Specification Clarifications A16 and A17.
November 1998	-020	Updated Specification Change A1, Documentation Change A11, Erratum A44, Specification Change A6 and the Pentium II Processor Identification Information table. Added Erratum A65 and Documentation Change A18.
December 1998	-021	Updated Specification Change A1 and the Pentium II Processor Identification Information table. Added Erratum A66. Updated status for Errata A16 through A29, A31, A35 through A39, A42, A48, A54, A57, and A60. Changed affected steppings for Erratum A32.
January 1999	-022	Updated Specification Change A1 and the Pentium II Processor Identification Information table. Added Errata A67 through A69, and Documentation Change A19 through A21.
February 1999	-023	Updated Processor Identification Information table. Added Erratum A70.
March 1999	-024	Added Specification Change A8 and updated the Pentium II Processor Identification Information table. Added S-Spec definition. Removed Specification Changes, Specification Clarifications, and Document Changes that have been incorporated into the appropriate documentation. Renumbered remaining items.
April 1999	-025	Added Documentation Change A4 and updated the Pentium II Processor Identification Information table. Moved revised Mixed Steppings statement to the General Information section and

REVISION HISTORY

Date of Revision	Version	Description
		renumbered remaining items.
May 1999	-026	Removed Specification and Documentation Changes that have been incorporated into the appropriate documentation and renumbered remaining items. Added Specification Change A3. Updated Erratum A57 Plans status to “Fix.”
June 1999	-027	Added Erratum A71. Added Documentation Change A2. Added Specification Clarifications A2 and A3. Added Specification Change A4. Corrected Pentium II Processor Identification Information table, Note 10.
July 1999	-028	Added Erratum A72. Corrected Pentium II Processor Identification Information table, Note 10 and table references to that note. Corrections in the May 1999 version were incorrect.
August 1999	-029	Added Documentation Change A3. Updated the Pentium II Processor Identification Information table and added Note 22. Moved Identification Information into the General Information section. Updated Codes Used in Summary Table. Updated column heading in Errata, Documentation Changes, Specification Clarifications and Specification Changes tables.
October 1999	-030	Added Errata A73. Added ‘Brand Id’ to *Identification Information* table.
November 1999	-031	Updated references at the beginning of each section. Updated *Pentium® II Processor Identification Information* table. Added Errata A74 and A75. Added Documentation Change A4.
December 1999	-032	Added Errata A76. Added Documentation Change A5. Added Specification Clarification A4.
January 2000	-033	Added Errata A77-A78. Added Documentation Change A6.
February 2000	-034	Updated Erratum A75. Added Documentation Change A7. Updated Summary of Changes product letter codes.
March 2000	-035	Updated Erratum A74.
May 2000	-036	Added Erratum A79 & A80.
September 2000	-037	Added New Errata A81, A82, A83, A84, A85. Added Errata Re-Writes A58, A69, A74, A78. Added Document Changes A8, A9.
October 2000	-038	Added New Erratum A86. .Added Document Changes A10, A11.
November 2000	-039	Added New Erratum A87.
December 2000	-040	Updated Specification Update product key to include the Intel® Pentium® 4 processor, Revised Erratum A2. Added Documentation Changes A12 - A17.

PREFACE

This document is an update to the specifications contained in the following documents:

- *P6 Family of Processors Hardware Developer's Manual* (Order Number 244001)
- *Pentium® II Processor Developer's Manual* (Order Number 243341)
- *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet (Order Number 243335)
- *Pentium® II Processor at 350 MHz, 400 MHz, and 450 MHz* datasheet (Order Number 243657)
- *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3* (Order Numbers 243190, 243191, and 243192, respectively)

It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains S-Specs, Errata, Documentation Changes, Specification Clarifications and, Specification Changes.

Nomenclature

S-Spec Number is a five-digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc. as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

Errata are design defects or errors. Errata may cause the Pentium II processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

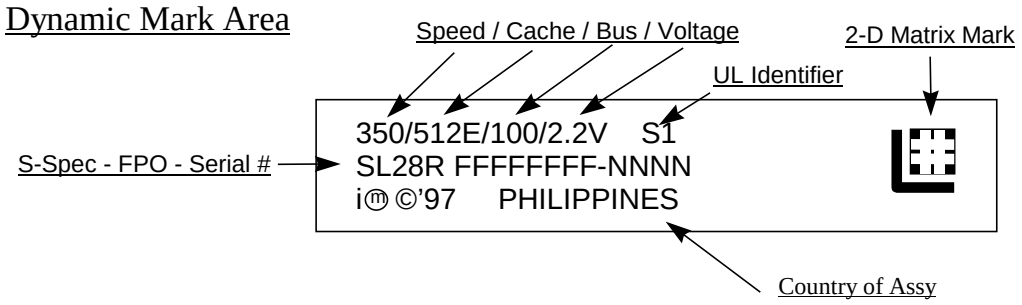
Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Specification Changes are modifications to the current published specifications for the Mobile Pentium® II processor or the Intel® Pentium® II Processor Mobile Module. These changes will be incorporated in the next release of the specifications.

Specification Update for the Pentium® II Processor

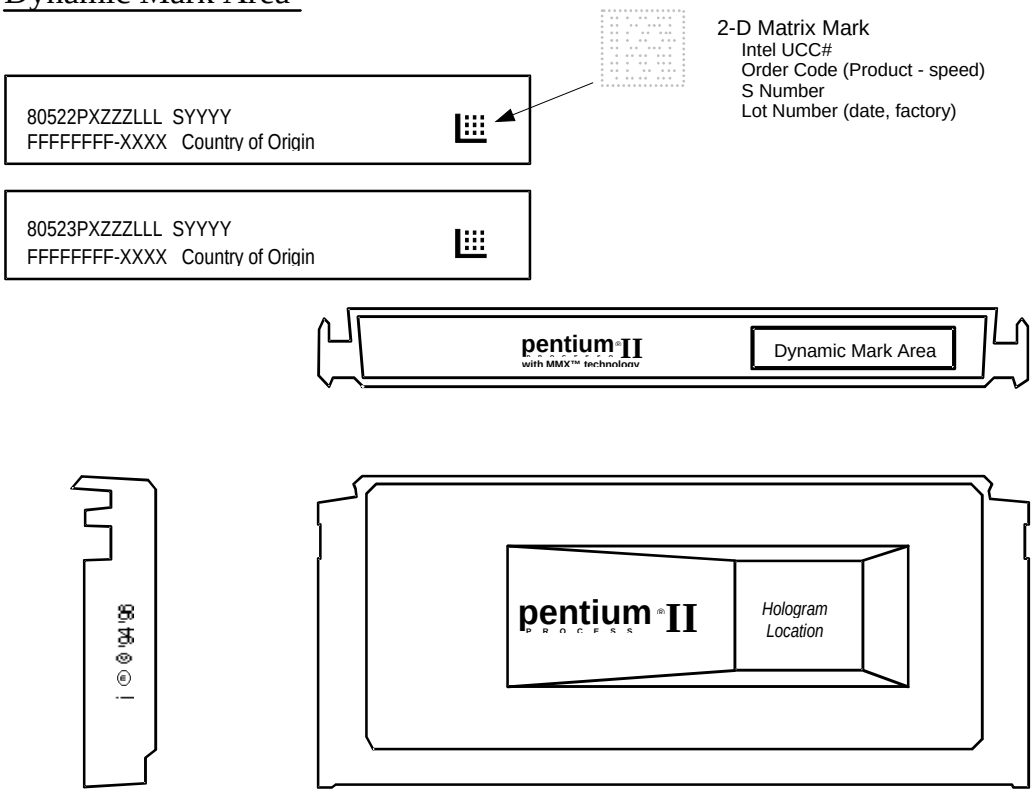
GENERAL INFORMATION

Pentium® II Processor and Boxed Pentium® II Processor 3 Line Markings



Pentium® II Processor Markings

Dynamic Mark Area

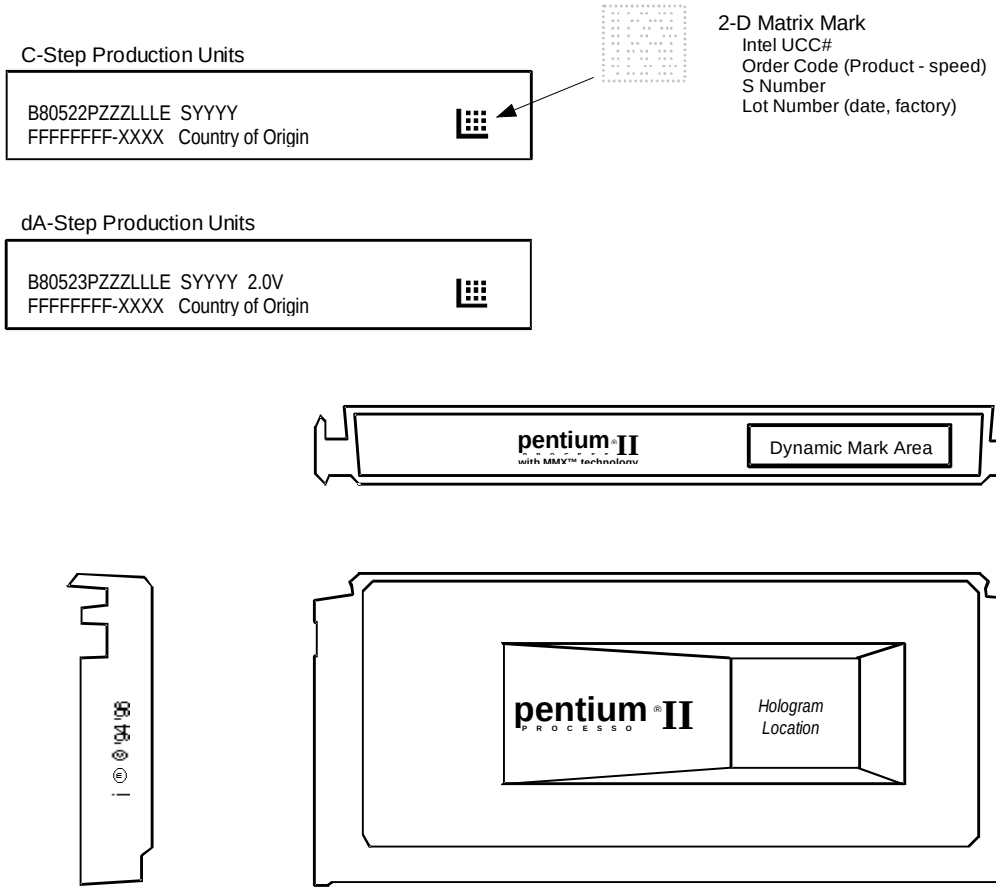


NOTES:

- ZZZ = Speed (MHz).
- SYYYY = S-spec Number.
- LLL = Level 2 Cache Size (in Kilobytes).
- FFFFFFFF = FPO # (Test Lot Traceability #).
- XXXX = Serialization Code.

Boxed Pentium® II Processor Markings

Dynamic Mark Area

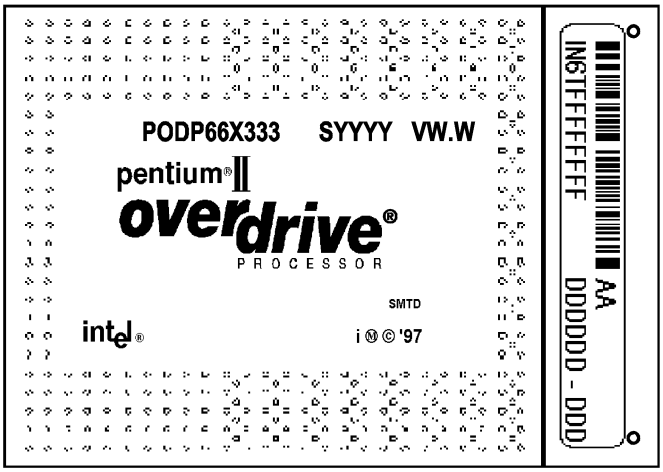


NOTES:

- ZZZ = Speed (MHz).
- LLL = Level 2 Cache Size (in Kilobytes).
- E = ECC Support in Level 2 Cache
- SYYYY = S-spec Number.
- FFFFFFFF = FPO # (Test Lot Traceability #).
- XXXX = Serialization Code.

Pentium® II OverDrive® Processor Line Markings

Bottom View of Pentium® II OverDrive® Processor



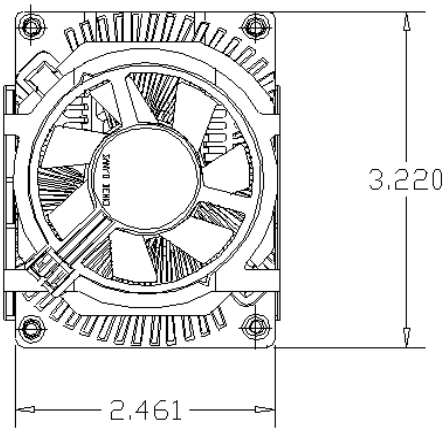
NOTES:

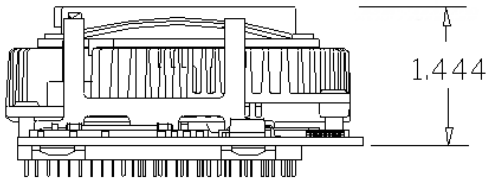
Label Markings

- FFFFFFF = FPO # (Test Lot Traceability #).
- DDDDDD – DDD = Altered Assembly Number.

Bottom Cover Markings

- PODP66X333 = Product Code.
- SYYYY = S-spec Number.
- VW.W = Version Number.





NOTES:

1. Attached fan heat sink is not end user removable.
2. Fan power is provided through external fan power connector, not through the processor socket.

IDENTIFICATION INFORMATION

The Pentium II processor can be identified by the following values:

Family[1]	233-, 266-, 300-, 333[3]-MHz Model 3[2]	266-, 300-, 333-, 350-, 400-, and 450-MHz Model 5[2]	Brand ID[4]
0110	0011	0101	00h = Not Supported

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.
3. This is a Pentium® II OverDrive® processor. Please note that although this processor has a CPUID of 163xh, it uses a Pentium II processor CPUID 065xh processor core.
4. The Brand ID corresponds to bits [7:0] of the EBX register after the CPUID instruction is executed with a 1 in the EAX register.

The Pentium II processor's second level (L2) cache size can be determined by the following register contents:

512-Kbyte Unified L2 Cache[1]	43h

NOTES:

1 For the Pentium® II processor, the unified L2 cache size corresponds to the value in bits [3:0] of the EDX register after the CPUID instruction is executed with a 2 in the EAX register. Other Intel microprocessor models or families may move this information to other bit positions or otherwise reformat the result returned by this instruction; generic code should parse the resulting token stream according to the definition of the CPUID instruction.

Mixed Steppings in DP Systems

Intel Corporation fully supports mixed steppings of Pentium II processors. The following list and processor matrix describes the requirements to support mixed steppings:

- While Intel has done nothing to specifically prevent processors operating at differing frequencies from functioning within a dual processor system, there may be uncharacterized errata that exist in such configurations. Intel does not support such configurations. In mixed stepping systems, all processors must operate at identical frequencies (i.e., the highest frequency rating commonly supported by all processors).
- While there are no known issues associated with the mixing of processors with differing cache sizes in a dual processor system, and Intel has done nothing to specifically prevent such system configurations from operating, Intel does not support such configurations since there may be uncharacterized errata that exist. In mixed stepping systems, all processors must be of the same cache size.
- While Intel believes that certain customers may wish to perform validation of system configurations with mixed frequency or cache sizes, and that those efforts are an acceptable option to our customers, customers would be fully responsible for the validation of such configurations.
- The workarounds identified in this and following specification updates must be properly applied to each processor in the system. Certain errata are specific to the multiprocessor environment and are identified in the *Mixed Stepping Processor Matrix* found at the end of this section. Errata for all processor steppings will affect system performance if not properly worked around. Also see the "Pentium® II Processor Identification and Package Information" table for additional details on which processors are affected by specific errata.
- In mixed stepping systems, the processor with the lowest feature-set, as determined by the CPUID Feature Bytes, must be the Bootstrap Processor (BSP). In the event of a tie in feature-set, the tie should be resolved by selecting the BSP as the processor with the lowest stepping as determined by the CPUID instruction.

In the following processor matrix, "NI" indicates that there are currently no known issues associated with mixing these steppings. A number indicates that a known issue has been identified as listed in the table following the matrix. A dual processor system using mixed processor steppings must assure that errata are addressed appropriately for each processor.

DP Platform Population Matrix for the Pentium® II Processor with 66 MHz System Bus

Pentium® II Processor Stepping	266 MHz C0	300 MHz C0	233 MHz C1	266 MHz C1	300 MHz C1	266 MHz dA0	333 MHz dA0	300 MHz dA1	333 MHz dA1	266 MHz dB0	300 MHz dB0	333 MHz dB0
266-MHz C0	**1**	X	X	**1**	X	**1**	X	X	X	**1**	X	X
300-MHz C0	X	**1**	X	X	**1**	X	X	**1**	X	X	**1**	X
233-MHz C1	X	X	**NI**	X	X	X	X	X	X	X	X	X
266-MHz C1	**1**	X	X	**NI**	X	**NI**	X	X	X	**NI**	X	X
300-MHz C1	X	**1**	X	X	**NI**	X	X	**NI**	X	X	**NI**	X
266-MHz dA0	**1**	X	X	**NI**	X	**NI**	X	X	X	**NI**	X	X
333-MHz dA0	X	X	X	X	X	X	**NI**	X	**NI**	X	X	**NI**
300-MHz dA1	X	**1**	X	X	**NI**	X	X	**NI**	X	X	**NI**	X
333-MHz dA1	X	X	X	X	X	X	**NI**	X	**NI**	X	X	**NI**
266-MHz dB0	**1**	X	X	**NI**	X	**NI**	X	X	X	**NI**	X	X

DP Platform Population Matrix for the Pentium® II Processor with 66 MHz System Bus

Pentium® II Processor Stepping	266 MHz C0	300 MHz C0	233 MHz C1	266 MHz C1	300 MHz C1	266 MHz dA0	333 MHz dA0	300 MHz dA1	333 MHz dA1	266 MHz dB0	300 MHz dB0	333 MHz dB0
300-MHz dB0	X	**1**	X	X	**NI**	X	X	**NI**	X	X	**NI**	X
333-MHz dB0	X	X	X	X	X	X	**NI**	X	**NI**	X	X	**NI**

NOTES:

1. Errata A16 and A17, as listed in the *Pentium® II Processor Specification Update*, may be problematic for DP systems that use Pentium® II processor, model 3 C0 stepping. Please see the *Pentium® II Processor Specification Update* for further information.

X = Mixing processors at different frequencies is not supported.

NI = No known issues associated with mixing these steppings.

DP Platform Population Matrix for the Pentium® II Processor with 100 MHz System Bus

Pentium® II Processor Stepping	350 MHz dA0	350 MHz dA1	400 MHz dA1	350 MHz dB0	400 MHz dB0	450 MHz dB0	350 MHz dB1	400 MHz dB1
350-MHz dA0	**NI**	**NI**	X	**NI**	X	X	**NI**	X
350-MHz dA1	**NI**	**NI**	X	**NI**	X	X	**NI**	X
400-MHz dA1	X	X	**NI**	X	**NI**	X	X	**NI**
350-MHz dB0	**NI**	**NI**	X	**NI**	X	X	**NI**	X
400-MHz dB0	X	X	**NI**	X	**NI**	X	X	**NI**
450-MHz dB0	X	X	X	X	X	**NI**	X	X
350-MHz dB1	**NI**	**NI**	X	**NI**	X	X	**NI**	X
400-MHz dB1	X	X	**NI**	X	**NI**	X	X	**NI**

NOTE:

X = Mixing processors at different frequencies is not supported.

NI = No known issues associated with mixing these steppings.

Pentium® II Processor Identification Information

S-Spec	Core Steppings	CPUID	Speed (MHz) Core/Bus	L2 Size (Kbytes)	TagRAM/ Stepping	ECC/Non -ECC	Processor Substrate Revision	Package and Revision	Notes
SL264	C0	0633h	233/66	512	T6/B0	non-ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL265	C0	0633h	266/66	512	T6/B0	non-ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL268	C0	0633h	233/66	512	T6/B0	ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL269	C0	0633h	266/66	512	T6/B0	ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL28K	C0	0633h	233/66	512	T6/B0	non-ECC	D	SECC 3.00	1, 2, 3, 9, 13, 20, 21
SL28L	C0	0633h	266/66	512	T6/B0	non-ECC	D	SECC 3.00	1, 2, 3, 9, 13, 20, 21
SL28R	C0	0633h	300/66	512	T6/B0	ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL2MZ	C0	0633h	300/66	512	T6/B0	ECC	D	SECC 3.00	1, 2, 3, 13, 20, 21
SL2HA	C1	0634h	300/66	512	T6/B0	ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL2HC	C1	0634h	266/66	512	T6/B0	non-ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL2HD	C1	0634h	233/66	512	T6/B0	non-ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL2HE	C1	0634h	266/66	512	T6/B0	ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL2HF	C1	0634h	233/66	512	T6/B0	ECC	D	SECC 3.00	1, 2, 13, 20, 21
SL2QA	C1	0634h	233/66	512	T6/B0	non-ECC	D	SECC 3.00	1, 2, 3, 9, 13, 20, 21
SL2QB	C1	0634h	266/66	512	T6/B0	non-ECC	D	SECC 3.00	1, 2, 3, 9, 13, 20, 21
SL2QC	C1	0634h	300/66	512	T6/B0	ECC	D	SECC 3.00	1, 2, 3, 13, 20, 21
SL2KA	dA0	0650h	333/66	512	T6P/A3	ECC	B1	SECC 3.00	4, 5, 8, 14, (20 or 21)
SL2QF	dA0	0650h	333/66	512	T6P/A3	ECC	B1	SECC 3.00	3, 4, 5, 8, 14
SL2K9	dA0	0650h	266/66	512	T6P/A3	ECC	B1	SECC 3.00	4, 5, 8, 14, 21
SL35V	dA1	0651h	300/66	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 5, 7, 8, 15
SL2QH	dA1	0651h	333/66	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 5, 7, 8, 15
SL2S5	dA1	0651h	333/66	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 5, 7, 8, 15, (20 or 21)

Pentium® II Processor Identification Information

S-Spec	Core Steppings	CPUID	Speed (MHz) Core/Bus	L2 Size (Kbytes)	TagRAM/ Stepping	ECC/Non -ECC	Processor Substrate Revision	Package and Revision	Notes
SL2ZP	dA1	0651h	333/66	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 5, 7, 8, 15, 19, 20
SL2ZQ	dA1	0651h	350/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 5, 7, 8, 15, 19, 20
SL2S6	dA1	0651h	350/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 6, 7, 8, 15, (20 or 21), 22
SL2S7	dA1	0651h	400/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 6, 7, 8, 10, 15, (20 or 21)
SL2SF	dA1	0651h	350/100	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 6, 7, 8, 15
SL2SH	dA1	0651h	400/100	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 6, 7, 8, 10, 15
SL2VY	dA1	0651h	300/66	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 6, 7, 8, 15
SL33D	dB0	0652h	266/66	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 5, 7, 8, 15, 20
SL2YK	dB0	0652h	300/66	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 5, 7, 8, 15, 20
SL2WZ	dB0	0652h	350/100	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 6, 7, 8, 15, 20
SL2YM	dB0	0652h	400/100	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 6, 7, 8, 10, 15, 20
SL37G	dB0	0652h	400/100	512	T6P-e/A0	ECC	B1	SECC2 OLGA	3, 7, 10, 12, 15, 18
SL2WB	dB0	0652h	450/100	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 7, 8, 10, 11, 15, 20
SL37H	dB0	0652h	450/100	512	T6P-e/A0	ECC	B1	SECC2 OLGA	3, 4, 7, 8, 10, 15, 18
SL2KE	TdB0	1632h	333/66	512	C6C/A3	ECC	N/A	PGA	4, 7, 8, 12
SL2W7	dB0	0652h	266/66	512	T6P-e/A0	ECC	B1	SECC 2.00	4, 5, 7, 8, 15, 20
SL2W8	dB0	0652h	300/66	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 5, 7, 8, 15, 20
SL2TV	dB0	0652h	333/66	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 5, 7, 8, 15, 20
SL2U3	dB0	0652h	350/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 6, 7, 8, 15, 20
SL2U4	dB0	0652h	350/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 6, 7, 8, 15, 20

Pentium® II Processor Identification Information

S-Spec	Core Steppings	CPUID	Speed (MHz) Core/Bus	L2 Size (Kbytes)	TagRAM/ Stepping	ECC/Non -ECC	Processor Substrate Revision	Package and Revision	Notes
SL2U5	dB0	0652h	400/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 6, 7, 8, 10, 15, 20, 22
SL2U6	dB0	0652h	400/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 6, 7, 8, 10, 15, 20
SL2U7	dB0	0652h	450/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 7, 8, 10, 11, 15, 20
SL356	dB0	0652h	350/100	512	T6P-e/A0	ECC	B1	SECC2 PLGA	4, 7, 8, 10, 15, 16, 20, 22
SL357	dB0	0652h	400/100	512	T6P-e/A0	ECC	B1	SECC2 OLGA	4, 7, 8, 10, 15, 18, 20
SL358	dB0	0652h	450/100	512	T6P-e/A0	ECC	B1	SECC2 OLGA	4, 7, 8, 10, 15, 17, 18, 20
SL37F	dB0	0652h	350/100	512	T6P-e/A0	ECC	B1	SECC2 PLGA	3, 4, 7, 8, 10, 15, 16, 20
SL3FN	dB0	0652h	350/100	512	T6P-e/0	ECC	B1	SECC2 OLGA	4, 7, 8, 10, 15, 18, 20
SL3EE	dB0	0652h	400/100	512	T6P-e/0	ECC	B1	SECC2 PLGA	1, 7, 8, 15, 16, 20, 22
SL3F9	dB0	0652h	400/100	512	T6Pe/A0	ECC	B1	SECC2 PLGA	3, 4, 7, 8, 10, 15, 16
SL38M	dB1	0653h	350/100	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 6, 7, 8, 10, 15, 20
SL38N	dB1	0653h	400/100	512	T6P-e/A0	ECC	B1	SECC 3.00	3, 4, 6, 7, 8, 10, 15, 20
SL36U	dB1	0653h	350/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 6, 7, 8, 10, 15, 20
SL38Z	dB1	0653h	400/100	512	T6P-e/A0	ECC	B1	SECC 3.00	4, 6, 7, 8, 10, 15, 20
SL3D5	dB1	0653h	400/100	512	T6P-e/A0	ECC	B1	SECC2 OLGA	3, 7, 8, 10, 15, 18
SL3J2	dB1	0653h	350/100	512	T6P-e/A0	ECC	B1	SECC2 PLGA	4, 7, 8, 10, 15, 16, 22

NOTES:

1. V_{CC_CORE} is specified for 2.8 V +100/-70 mV for all Pentium® II processors.
2. T_{PLATE} is specified for 5° C – 75° C for these Pentium II processors with S.E.C. cartridge packages except for s-specs SL28R , SL2HA, SL2MZ, and SL2QC which have a T_{PLATE} specification for 5º C – 72º C.
3. This is a boxed Pentium II processor with an attached fan heatsink.
4. $V_{CC_{CORE}}$ is specified for 2.0 V +100/-70 mV for these Pentium II processors.
5. T_{PLATE} is specified for 5° C – 65° C for these Pentium II processors.
6. T_{PLATE} is specified for 5° C – 75° C with ETP (extended thermal plate) for these Pentium II processors.

7. Cacheable address space supports up to 4 Gbytes for these Pentium II processors.
8. These processors will not shut down automatically on THERMTRIP#.
9. These boxed processors may have packaging which incorrectly indicates ECC support in the L2 cache.
10. These processors are affected by Erratum A56.
11. T_{PLATE} is specified for 5° C – 70° C with ETP (extended thermal plate) for these Pentium II processors.
12. This is a boxed Pentium II OverDrive® processor with an attached fan heatsink.
13. This TagRAM notation is equivalent to part number 82459AB.
14. This TagRAM notation is equivalent to part number 82459AC.
15. This TagRAM notation is equivalent to part number 82459AD.
16. T_{CASE} (MAX) is specified as 80° C for these Pentium II processors.
17. These processors are affected by Erratum A67.
18. $T_{JUNCTION\ (MAX)}$ is specified as 90° C for these Pentium II processors.
19. These processors require a dual reset BIOS.
20. These parts will only operate at the specified core to bus frequency ratio at which they were manufactured and tested. It is not necessary to configure the core frequency ratios by using the A20M#, IGNEE#, LINT[1]/NMI and LINT[0]/INTR pins during RESET.
21. These parts require the inputs from A20M#, IGNEE#, LINT[1]/NMI and LINT[0]/INTR pins during RESET to set the correct core to bus frequency ratio.
22. This part also ships as a boxed processor with an attached fan heatsink.

SUMMARY OF CHANGES

The following table indicates the Errata, Documentation Changes, Specification Clarifications, or Specification Changes that apply to Pentium® II processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Erratum, Documentation Change, Specification Clarification or Specification Change applies to the given processor stepping.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
Doc:	Intel intends to update the appropriate documentation in a future revision.
PKG:	This column refers to errata on the Pentium II processor substrate.
AP:	APIC related erratum.
Shaded:	This item is either new or modified from the previous version of the document.

Each Specification Update item is prefixed with a capital letter to distinguish the product. The key below details the letters that are used in Intel's microprocessor Specification Updates:

A = Intel® Pentium® II processor

B = Intel® Mobile Pentium® II processor

C = Intel® Celeron™ processor

D = Intel® Pentium® II Xeon™ processor

E = Intel® Pentium® III processor

G = Intel® Pentium® III Xeon™ processor

H = Intel® Mobile Celeron™ processor at 466 MHz, 433 MHz, 400 MHz, 366 MHz, 333 MHz, 300 MHz, and 266 MHz

K = Intel® Mobile Pentium® III processor

M = Intel® Mobile Celeron™ processor at 500 MHz, 450 MHz, 400A MHz

N = Intel® Pentium® 4 processor

The Specification Updates for the Pentium® processor, Pentium® Pro processor, and other Intel products do not use this convention.

Summary of Errata

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
A1	X	X	X	X	X	X	X		NoFix	FP data operand pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
A2	X	X	X	X	X	X	X		NoFix	Differences exist in debug exception reporting
A3	X	X	X	X	X	X	X		NoFix	FLUSH# servicing delayed while waiting for STARTUP_IPI in 2-way MP systems
A4	X	X	X	X	X	X	X		NoFix	Code fetch matching disabled debug register may cause debug exception
A5	X	X	X	X	X	X	X		NoFix	Double ECC error on read may result in BINIT#
A6	X	X	X	X	X	X	X		NoFix	FP inexact-result exception flag may not be set
A7	X	X	X	X	X	X	X		NoFix	BTM for SMI will contain incorrect FROM EIP
A8	X	X	X	X	X	X	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
A9	X	X	X	X	X	X	X		NoFix	Branch traps do not function if BTMs are also enabled
A10	X	X	X	X	X	X	X		NoFix	Checker BIST failure in FRC mode not signaled
A11	X	X	X	X	X	X	X		NoFix	BINIT# assertion causes FRCERR assertion in FRC mode
A12	X	X	X	X	X	X	X		NoFix	Machine check exception handler may not always execute successfully
A13	X	X	X	X	X	X	X		NoFix	MCE due to L2 parity error gives L1

Summary of Errata

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
										MCACOD.LL
A14	X	X	X	X	X	X	X		NoFix	LBER may be corrupted after some events
A15	X	X	X	X	X	X	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
A16	X								Fixed	System may hang due to internal protocol violation
A17	X								Fixed	Livelock condition may cause system hang
A18	X	X							Fixed	Mispredicted branch may cause incorrect tag word on MMX™ technology instructions
A19	X	X							Fixed	Thermal sensor/THERMTRIP# does not work
A20	X	X							Fixed	Spurious machine check exception via IFU data parity error
A21	X	X							Fixed	Loss of inclusion in IFU can cause machine check exception
A22	X	X							Fixed	Possible system hang when paging is disabled and reenabled from uncached memory
A23	X	X							Fixed	L2 performance counters miscount L2_RQSTS
A24	X	X							Fixed	Erroneous signaling of user mode protection violation
A25	X								Fixed	Invalid operation not signaled by the FIST instruction on some out of range operands
A26	X	X							Fixed	FLUSH# assertion disables L2 machine check exception reporting
A27	X	X							Fixed	EFLAGS may be

Summary of Errata

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
										incorrect after a multiprocessor TLB shootdown
A28	X	X	X	X					Fixed	Delayed line invalidation issue during 2-way MP data ownership transfer
A29	X	X	X	X					Fixed	Potential early deassertion of LOCK# during split-lock cycles
A30	X	X	X	X	X	X	X		NoFix	A20M# may be inverted after returning from SMM and Reset
A31	X	X	X	X					Fixed	Reporting of floating-point exception may be delayed
A32	X	X	X	X	X	X	X		NoFix	EFLAGS discrepancy on a page fault after a multiprocessor TLB shootdown
A33	X	X	X	X	X	X	X		NoFix	Near CALL to ESP creates unexpected EIP address
A34									Fixed	Deep sleep exit transition may cause hang
A35			X	X					Fixed	Built-in self test always gives nonzero result
A36			X	X					Fixed	THERMTRIP# may not be asserted as specified
A37			X						Fixed	Cache state corruption in the presence of page A/D-bit setting and snoop traffic
A38			X						Fixed	Snoop cycle generates spurious machine check exception
A39	X	X	X	X					Fixed	MOVD/MOVQ instruction writes to memory prematurely
A40	X	X	X	X	X	X	X		NoFix	Memory type undefined for nonmemory operations

Summary of Errata

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
A41			X	X	X	X	X		NoFix	Infinite snoop stall during L2 initialization of MP systems
A42	X	X	X	X					Fixed	Bus protocol conflict with optimized chipsets
A43	X	X	X	X	X	X	X		NoFix	FP data operand pointer may not be zero after power on or Reset
A44	X	X	X	X	X	X	X		NoFix	MOVD following zeroing instruction can cause incorrect result
A45	X	X	X	X	X	X	X		NoFix	Premature execution of a load operation prior to exception handler invocation
A46	X	X	X	X	X	X	X		NoFix	Read portion of RMW instruction may execute twice
A47	X	X	X	X	X	X	X		Fix	Test pin must be high during power up
A48	X	X	X	X	X	X	X		NoFix	Intervening writeback may occur during locked transaction
A49	X	X	X	X	X	X	X		NoFix	MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed
A50	X	X	X	X	X	X	X		NoFix	Mixed cacheability of lock variables is problematic in MP systems
A51	X	X	X	X	X	X	X		NoFix	MOV with debug register causes debug exception
A52			X	X	X	X	X		NoFix	Upper four PAT entries not usable with Mode B or Mode C paging
A53	X	X	X	X	X	X	X		Fix	UC write may be reordered around a cacheable write
A54			X	X					Fixed	Incorrect memory type may be used when MTRRs are disabled

Summary of Errata

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
A55	X	X	X	X	X	X	X		Fix	Misprediction in program flow may cause unexpected instruction execution
A56			X	X	X	X	X		Fix	System bus ECC may report false errors
A57	X	X	X	X	X	X			Fix	Full in-order queue may cause infinite DBSY# assertion
A58	X	X	X	X	X	X	X		NoFix	Data breakpoint exception in a displacement relative near call may corrupt EIP
A59			X	X	X	X	X		NoFix	System bus ECC not functional with 2:1 ratio
A60	X	X	X	X	X	X	X		NoFix	Fault on REP CMPS/SCAS operation may cause incorrect EIP
A61	X	X	X	X	X	X	X		NoFix	RDMSR and WRMSR to invalid MSR may not cause GP fault
A62	X	X	X	X	X	X	X		NoFix	SYSENTER/SYSEXIT instructions can implicitly load “null segment selector” to SS and CS registers
A63	X	X	X	X	X	X	X		NoFix	PRELOAD followed by EXTEST does not load boundary scan data
A64	X	X	X	X	X	X	X		NoFix	Far jump to new TSS with D-bit cleared may cause system hang
A65	X	X	X	X	X	X	X		Fix	Incorrect chunk ordering may prevent execution of the machine check exception handler after BINIT#
A66	X	X	X	X	X	X	X		NoFix	Resume Flag may not be cleared after debug exception
A67	X	X	X	X	X	X	X		NoFix	System bus address

Summary of Errata

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
										parity generator may report false AERR#s
A68	X	X	X	X	X	X	X		NoFix	Misaligned locked access to APIC space results in hang
A69	X	X	X	X	X	X	X		NoFix	Potential loss of data coherency during MP data ownership transfer
A70	X	X	X	X	X	X	X		NoFix	Memory ordering based synchronization may cause a livelock condition in MP systems
A71	X	X	X	X	X	X	X		NoFix	GP# fault on WRMSR to ROB_CR_BKUPTMPDR6
A72	X	X	X	X	X	X	X		NoFix	Machine check exception may occur due to improper line eviction in the IFU
A73	X	X	X	X	X	X	X		NoFix	Lower bits of SMRAM SMBASE register cannot be written with an ITP
A74	X	X	X	X	X	X	X		NoFix	Task switch may cause wrong PTE and PDE access bit to be set
A75	X	X	X	X	X	X	X		NoFix	Unsynchronized Cross-Modifying code operations can cause unexpected instruction execution results
A76	X	X	X	X	X	X	X		NoFix	Deadlock may occur due to illegal-instruction/page-miss combination
A77	X	X	X	X	X	X	X		NoFix	FLUSH# assertion following STPCLK# may prevent CPU clocks from stopping
A78	X	X	X	X	X	X	X		NoFix	Floating-point exception condition may be deferred

Summary of Errata

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
A79	X	X	X	X	X	X	X		NoFix	Snoop probe during FLUSH# could cause L2 to be left in shared state
A80	X	X	X	X	X	X	X		NoFix	Livelock may occur due to IFU line eviction
A81	X	X	X	X	X	X	X		NoFix	Selector for the LTR/LLDT register may get corrupted
A82	X	X	X	X	X	X	X		NoFix	INIT does not clear global entries in the TLB
A83	X	X	X	X	X	X	X		NoFix	VM bit will be cleared on a double fault handler
A84	X	X	X	X	X	X	X		NoFix	Memory aliasing with inconsistent A and D bits may cause processor deadlock
A85	X	X	X	X	X	X	X		NoFix	Use of memory aliasing with inconsistent memory type may cause system hang
A86	X	X	X	X	X	X	X		NoFix	Processor may report invalid TSS fault instead of double fault during mode C paging
A87	X	X	X	X	X	X	X		NoFix	Machine check exception may occur when interleaving code between different memory types
A1AP	X	X	X	X	X	X	X		NoFix	APIC access to cacheable memory causes SHUTDOWN
A2AP	X	X	X	X	X	X	X		NoFix	2-way MP systems may hang due to catastrophic errors during BSP determination
A3AP	X	X	X	X	X	X	X		NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt

Summary of Documentation Changes

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
A1					X				Doc	400 MHz S.E.C.C.2 PLGA part spec addition
A2	X	X	X	X	X	X	X		Doc	STPCLK# pin definition
A3	X	X	X	X	X	X	X		Doc	Invalidating caches and TLBs
A4	X	X	X	X	X	X	X		Doc	Handling of self-modifying and cross-modifying code
A5	X	X	X	X	X	X	X		Doc	Machine check architecture initialization of MCi_STATUS registers
A6	X	X	X	X	X	X	X		Doc	LOCK# signal prefix operands
A7	X	X	X	X	X	X	X		Doc	SMRAM state save map contains documentation errors
A8	X	X	X	X	X	X	X		Doc	Memory Aliasing with different memory types
A9	X	X	X	X	X	X	X		Doc	System Management Interrupt (SMI) during start-up IPI Clarification
A10	X	X	X	X	X	X	X		Doc	Runbist will not function when STPCLK# driven low
A11	X	X	X	X	X	X	X		Doc	Memory aliasing with inconsistent A & D bits may cause processor deadlock
A12	X	X	X	X	X	X	X		Doc	An Interrupt may occur while TSS is marked busy
A13	X	X	X	X	X	X	X		Doc	NMI unmasked early when processor is running in V86 mode
A14	X	X	X	X	X	X	X		Doc	P6 reads two bytes for POP SEG instruction
A15	X	X	X	X	X	X	X		Doc	APIC register offsets are aligned on 128 bit boundaries

Summary of Documentation Changes

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
A16	X	X	X	X	X	X	X		Doc	Single stepping of instructions breaks out of HLT state
A17	X	X	X	X	X	X	X		Doc	Additional signal resumes execution while in a HALT state

Summary of Specification Clarifications

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
A1			X	X	X				Doc	PWRGOOD inactive pulse width
A2									Doc	MTRR initialization clarification
A3									Doc	Floating-point opcode clarification
A4	X	X	X	X	X	X	X		Doc	Non-AGTL+ output low current clarification

Summary of Specification Changes

NO.	C0	C1	dA0	dA1	dB0	TdB0	dB1	PKG	Plans	ERRATA
A1	X	X	X	X	X	X	X		Doc	FRCERR pin removed from specification
A2	X	X	X	X	X	X	X		Doc	Non-GTL+ output leakage current change
A3	X	X	X	X	X	X	X		Doc	350 MHz, 400 MHz and 450 MHz Tjunc offset temperature spec addition
A4	X	X	X	X	X	X	X		Doc	RESET# pin definition

ERRATA

A1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

Problem: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

Implication: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating-point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

Workaround: If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A2. Differences Exist in Debug Exception Reporting

Problem: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Pentium III processor, as described below:

Case 1: The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Pentium III processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

Case 2: In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which:

a) crosses a 4-Kbyte page boundary,

OR

b) causes the page table Access or Dirty (A/D) bits to be modified,

the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information under these boundary conditions.

Case 3: If they occur after a MOVSS or POPSS instruction, the INT*n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

Case 4: If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

Case 5: When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

Implication: When debugging or when developing debuggers for a Pentium III processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.

Workaround: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4 or 5.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A3. FLUSH# Servicing Delayed While Waiting for STARTUP_IPI in 2-way MP Systems

Problem: In a 2-way MP system, if an application processor is waiting for a startup inter-processor interrupt (STARTUP_IPI), then it will not service a FLUSH# pin assertion until it has received the STARTUP_IPI.

Implication: After the 2-way MP initialization protocol, only one processor becomes the bootstrap processor (BSP). The other processor becomes a slave application processor (AP). After losing the BSP arbitration, the AP goes into a wait loop, waiting for a STARTUP_IPI.

The BSP can wake up the AP to perform some tasks with a STARTUP_IPI, and then put it back to sleep with an initialization inter-processor interrupt (INIT_IPI, which has the same effect as asserting INIT#), which returns it to a wait loop. The result is a possible loss of cache coherency if the off-line processor is intended to service a FLUSH# assertion at this point. The FLUSH# will be serviced as soon as the processor is awakened by a STARTUP_IPI, before any other instructions are executed. Intel has not encountered any operating systems that are affected by this erratum.

Workaround: Operating system developers should take care to execute a WBINVD instruction before the AP is taken off-line using an INIT_IPI.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A4. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

Problem: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

Implication: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

Workaround: The debug handler should clear breakpoint registers before they become disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A5. Double ECC Error on Read May Result in BINIT#

Problem: For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the Pentium II processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

Implication: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

Workaround: Though the ability to drive BINIT# can be disabled in the Pentium II processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A6. FP Inexact-Result Exception Flag May Not Be Set

Problem: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

Implication: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a “sticky” bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

Workaround: This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A7. BTM for SMI Will Contain Incorrect FROM EIP

Problem: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

Implication: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A8. I/O Restart in SMM May Fail After Simultaneous MCE

Problem: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Pentium II processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

Implication: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

Workaround: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A9. Branch Traps Do Not Function If BTMs Are Also Enabled

Problem: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

Implication: The branch traps and branch trace message debugging features cannot be used together.

Workaround: If branch trap functionality is desired, BTMs must be disabled.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A10. Checker BIST Failure in FRC Mode Not Signaled

Problem: If a system is running in functional redundancy checking (FRC) mode, and the checker of the master-checker pair encounters a hard failure while running the built-in self test (BIST), the checker will tri-state all outputs without signaling an IERR#.

Implication: Assuming the master passes BIST successfully, it will continue execution unchecked, operating without functional redundancy. However, the necessary pull-up on the FRCERR pin will cause an FRCERR to be signaled. The operation of the master depends on the implementation of FRCERR.

Workaround: For successful detection of BIST failure in the checker of an FRC pair, use the FRCERR signal, instead of IERR#.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A11. BINIT# Assertion Causes FRCERR Assertion in FRC Mode

Problem: If a pair of Pentium II processors are running in functional redundancy checking (FRC) mode, and a catastrophic error condition causes BINIT# to be asserted, the checker in the master-checker pair will enter shutdown. The next bus transaction from the master will then result in the assertion of FRCERR.

Implication: Bus initialization via an assertion of BINIT# occurs as the result of a catastrophic error condition which precludes the continuing reliable execution of the system. Under normal circumstances, the master-checker pair would remain synchronized in the execution of the BINIT# handler. However, due to this erratum, an FRCERR will be signaled. System behavior then depends on the system specific error recovery mechanisms.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A12. Machine Check Exception Handler May Not Always Execute Successfully

Problem: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

Implication: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A13. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

Problem: If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the Pentium II processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note that L2 ECC errors have the correct value of '10' logged.

Implication: An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A14. LBER May Be Corrupted After Some Events

Problem: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the re-initialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

Implication: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A15. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

Problem: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

Implication: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A16. System May Hang Due To Internal Protocol Violation

Problem: Pentium II processor-based systems may hang due to an internal protocol violation. When a snoopable transaction is issued on the bus and the cache line being accessed is in the modified state, the processor must deliver to the system bus an updated copy of the cache line. When the processor attempts to deliver the most up to date copy via an implicit writeback, the data transfer transaction fails and the DBSY# signal remains asserted until the next RESET#. This causes the system to hang indefinitely. In order to encounter this erratum, the following sequence of events must occur:

1. A snoopable transaction (transaction 1) is issued on the system bus. The processor contains in its L1 and/or L2 caches the data for this line in the modified state.
2. Another snoopable transaction (transaction 2) is issued and the processor contains this line only in its L2 cache in the modified state. Both of these transactions can be issued by either the chipset, by the processor (in which case they are of the self-snoop type), by another processor (2-way MP systems), or any combination thereof.
3. A nonsnoopable transaction is then issued (transaction 3) for which address bits A15-A5 are the same as those in transaction 2.
4. Transaction 3 is followed by a snoopable transaction (transaction 4).
5. The completion of the data transfer phase of transaction 1 must line up with the snoop response phase of transaction 3. This data transfer phase of transaction 1 must occur after the ADS# of transaction 4 and line up with the completion of an internal cache transaction.
6. The internal cache transaction must miss the L2 targeting a line for eviction, but the internal cache transaction must be such that it has to be retried.

The result of this sequence of transactions causes the processor bus to lock up after delivering the data for transaction 1, but prior to delivering the data for transaction 2. Since this data is never delivered, DBSY# does not deassert and the system hangs.

Implication: The Pentium II processor may cause a system to hang if the above listed sequence of events occurs. This sequence is a necessary condition to hit the erratum, but multiple variations of this sequence which also cause this erratum are also possible. The probability of encountering this erratum increases with I/O queue depth greater than 4 and in 2-way MP systems.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A17. Livelock Condition May Cause System Hang

Problem: A “livelock” situation could occur in 2-way MP Pentium II processor-based systems, when IOQ depth is set to 1, with a failure signature such that a processor arbitrates for the system bus but fails to drive out a transaction when it gains ownership of the bus. The processor then relinquishes bus ownership to another requester, but on rearbitration performs the same repetitive actions. This course of action continues until RESET# is asserted. The failure signature in 2-way MP systems is such that both processors require execution of an explicit writeback cycle and both processors request the bus for this transaction. However, when the time comes to drive out the writeback transaction, the internal request has been suspended due to an internal blocking condition. After the internal blocking condition has gone away the original writeback request is reasserted. However, by the time bus ownership has been regained, the blocking condition has recurred, thus suppressing the writeback request before the transaction can be driven out to the system bus.

The writeback which is waiting to go out on the system bus must be issued before the internal blocking condition can be removed. But the writeback can never be issued because of the recurring blocking condition. This causes an “infinite loop” situation to develop, and the processor essentially stops executing code.

Implication: This erratum was observed to occur when both processors are configured for IOQ depth = 1 in Intel commercial system testing.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A18. Mispredicted Branch May Cause Incorrect Tag Word on MMX™ Technology Instructions

Problem: After any MMX™ technology instruction is executed, all of the FPU stack registers should be marked valid in the FPU tag word. If one or more of the first three instructions of a mispredicted branch are MMX technology instructions of the form “opcode reg, mem” not including MOVD and MOVQ, the FPU tag word is incorrectly modified. Some of the tag word bits may remain invalid. This tag word will remain incorrect until one of two events occur:

1. Any MMX technology instruction is executed four or more instructions after the branch target, or
2. An MMX technology instruction of the following type is executed:

- Any MMX technology instruction of the form “opcode reg, reg”
- MOVD
- MOVQ
- EMMS

The following are examples of code that will encounter this erratum.

Example 1:

```
        EMMS
        ...
        Jcc     target          ; mispredicted as not taken
        ...
target:
        PADDW   mm0, [edi]      ; Is an “reg, mem” format instruction
        FSTENV  env
```

In this example, the tag word stored in memory by FSTENV will be incorrect.

Example 2:

```
        EMMS
        ...
        Jcc     target          ; mispredicted as not taken
        ...
target:
        PADDW   mm0, [edi]
        FUCOMPP                 ; depends on tag word, also violates coding guideline against mixing
                                ; floating-point and MMX technology instructions
        FWAIT
```

In this example, the FUCOMPP instruction will cause a Numeric Invalid Operation Exception if the FPU stack fault exception is unmasked.

Implication: When writing code that mixes FP and MMX technology instructions where the target of a branch is an MMX technology instruction with a memory operand, the FPU tag word may be incorrect. Software that expects the FP stack register to be set to valid after an MMX technology instruction and utilizes this information may be affected.

If floating-point instructions are intermixed, the floating-point instructions may raise the floating-point stack exception. If this exception is unmasked, the application will receive an unexpected numeric exception. The result is application dependent. If the floating-point stack exception is masked, the floating-point instruction will compute with a indefinite operand instead of the register contents. In either case the result is application dependent. Applications that follow the Intel MMX Technology Coding Guidelines against intermixing floating-point and MMX technology code are not affected by this erratum.

If the floating-point tag word is saved immediately after an affected MMX technology instruction, an erroneous value will be stored. Program behavior is application dependent. This may also cause debuggers to temporarily display incorrect tag word contents.

Workaround: All of the following must be applied to work around this erratum:

- Follow the Intel MMX technology guidelines in the *Intel Architecture Optimization Manual* for writing MMX technology programs. Specifically, do not intermix MMX technology instructions and floating-point instructions on a per instruction basis.
- If it is possible that some of the tag word bits may be invalid prior to a branch, avoid using MMX technology instructions of the form “opcode reg, mem”, except MOVD, MOVQ, within the first three instructions at the target of a branch.
- Use the FSAVE instruction to save all floating-point stack registers if at least one of the registers is valid during a context switch.
- Before a transition from MMX technology code to floating-point code that does not meet the Intel MMX Technology Guidelines in the *Intel Architecture Optimization Manual*, execute a nonsusceptible MMX technology instruction such as MOVD eax, mm0.

Floating-point instructions should not depend on MMX technology instructions to set the tag word bits to valid.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A19. Thermal Sensor/THERMTRIP# Does Not Work

Problem: THERMTRIP# is a feature of the Pentium II processor which asserts when the core reaches a certain temperature during operation as specified in the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet. The Pentium II processor may assert THERMTRIP# at a temperature lower or higher than the specified trippoint of 135° C for $T_{JUNCTION}$. When THERMTRIP# is asserted, the processor may shut down causing all execution to be halted.

Implication: When running the Pentium II processor, the Pentium II processor core may reach a temperature causing the processor to assert THERMTRIP# early. Once THERMTRIP# has been asserted, the processor may shut down due to this erratum. All execution after the SHUTDOWN will be halted. This erratum is only exhibited when T_{PLATE} is above the Maximum Specification of 75° C (see the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet (Order Number 243335) for details on specifications).

Workaround: Avoid operation of the Pentium II processor outside of thermal specifications defined by the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet. Do not monitor the THERMTRIP# pin (pin A15).

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A20. Spurious Machine Check Exception Via IFU Data Parity Error

Problem: The Pentium II processor can signal an unrecoverable Machine Check Exception (MCE) in the event that the Instruction Fetch Unit (IFU) detects a mismatch when verifying instruction parity. The execution of code which modifies the current instruction sequence that may already be fetched into the processor can cause an instruction at a given address to appear differently depending on when it was fetched in time relative to its being modified. Thus, a speculatively prefetched instruction may have been modified such that it now differs from the copy of the same instruction resident in the instruction cache. This discrepancy (of one copy located in the speculative prefetch portion, and a different copy in the instruction cache) is sensed by the IFU. When the IFU detects that the instruction stream has been modified, it flushes the pipeline and attempts to restart the instruction stream. In the interim, the IFU recognizes the disparate instructions described above, and signals a data parity error. The data parity error is signaled as an MCE before the instruction stream has had a chance to restart. This MCE will cause an operating system that has enabled MCE to shut down. No incorrect code is executed by the processor in this situation (even if MCE is disabled). Note that this erratum occurs under a specific set of address dependencies and timing events.

Implication: Executing such a sequence by modifying code without proper synchronization may not always result in predictable program behavior. The processor's signaling of an MCE due to a data parity error in the IFU may then result in an unexpected system halt if the above conditions are met and MCEs are enabled.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A21. Loss of Inclusion In IFU Can Cause Machine Check Exception

Problem: The Pentium II processor can signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism in the event that the Instruction Fetch Unit (IFU) detects differences in the consistency of code in instruction streaming buffers against code resident in the instruction cache, i.e., a loss of inclusion. When application code makes an operating system call, the processor transitions execution privilege levels. If the code for the OS call is not already resident in the level 1 cache, then the processor may prefetch code while identifying a cache line(s) for eventual eviction to make space for the new code. Upon return from the OS call, the processor continues execution of application code at the user level. The processor, due to deep speculation and branch prediction, may attempt to execute instructions from the previously prefetched kernel code starting by attempting to replace the victim line with kernel code in a buffer internal to the IFU. The IFU detects that the current application is insufficiently privileged to execute the kernel code and so, suppresses the eviction of the previously selected victim line. Despite having detected this condition, the IFU does replace this victim line with the kernel line. If the processor now attempts to restart execution of the current application code by refetching the original victim line it no longer finds it in the instruction cache. The IFU detects this loss of inclusion, and signals this by generating a MCE. If MCEs are enabled, this event can cause an operating system to shutdown. Note that this erratum occurs under a specific set of address dependencies and timing events.

Implication: The occurrence of all the conditions above can lead the IFU to signal a loss of inclusion by generating an MCE. If MCEs are enabled in the system, then the operating system may shut down upon noticing the MCE resulting in system failure. If MCEs are disabled, then unpredictable application behavior is theoretically possible, although current validation has shown execution to continue normally.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A22. Possible System Hang When Paging Is Disabled and Reenabled from Uncached Memory

Problem: If paging is disabled via the PG bit of CR0 and then later reenabled while executing code from a page marked uncachable by its Page Table Entry (PCD=1) but located in memory mapped as Write Back or Write Through by the processor MTRRs, the processor could internally enter a state resulting in a system hang.

Implication: Operating systems that enable and disable paging with the above described memory configurations could hang. Intel has not observed this erratum to date in laboratory testing of commercially available operating systems and applications.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A23. L2 Performance Counters Miscount L2_RQSTS

Problem: L2_RQSTS is a performance counter that counts the number of L2 cache access requests. This counter increments for each incoming L2 cache request. In some cases, an L2 request cannot be serviced by the L2 Cache. This request is then retried at a later time when the request can be serviced by the L2 cache. When this happens, the L2_RQSTS counter counts the initial L2 cache request *and* the retried L2 cache request, thereby counting the same request twice.

Implication: The L2_RQSTS counter may contain a larger erroneous number of L2 cache requests due to this erratum. This erratum does not affect functionality of the Pentium II processor. This erratum only affects the performance counter specified.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A24. Erroneous Signaling of User Mode Protection Violation

Problem: If the Pentium II processor attempts to access a page in physical memory marked not present (Present bit clear), a page fault exception (#PF) is generated. Before proceeding, there is a narrow internal timing window where the processor verifies that no other higher priority fault conditions are present. During this time, it is possible for another agent to allocate a new page directory or page table entry (PDE/PTE) corresponding to the same linear address of the original access, writing new values into the PDE/PTE with the Access bit (A-bit) or the Dirty bit (D-bit) cleared. When the original processor completes its checking for other fault conditions, and re-examines the A/D bit of the recently modified PDE/PTE, it finds that it has been cleared. Internal hardware correctly signals this scenario as a condition to which the processor should respond by setting the A/D bit, but erroneously reports it as a generic paging protection violation. Instead of attempting to set the appropriate A/D bit, this event is reported as an Int14 with exception code 0x05, i.e., user mode protection violation.

Implication: The occurrence of this scenario will result in the erroneous signaling of a user mode protection violation instead of a page fault and may result in application termination depending on operating system behavior in response to a user mode protection violation. Intel has only observed this erratum to date in laboratory testing of multi-processor systems.

Workaround: Operating systems which allocate new PTEs and PDEs should set the Access bit (A-bit) and Dirty bit (D-bit) to workaround this erratum. Alternatively, an operating system's Int14 handler can determine if a protection violation condition truly exists, and if none is found, return without further action.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A25. Invalid Operation Not Signaled by the FIST Instruction on Some Out of Range Operands

Problem: On certain, large, negative, floating-point operands, and only in three of the four possible processor rounding modes, the instructions FIST[P] m16int and FIST[P] m32int do not detect that the operand is so large that it will not fit into the target data size. As a consequence, the expected Invalid Operation exception response for this situation is not correctly provided, nor is the Invalid Operation flag set in the Floating-Point status word as specified in the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*. Under the failing conditions, noted below, the precision exception (#PE) flag will also be incorrectly set.
The erratum occurs only when all of the following conditions are met:

1. The FIST[P] instruction is either a 16- or 32-bit operation; 64-bit operations are unaffected.
2. Either the '*to nearest* ', '*to zero*' or '*up*' rounding modes are being used. The round '*down*' mode is unaffected by this erratum.
3. The sign bit of the floating-point operand is negative.
4. The floating-point operand being converted is significantly more negative than can be described by the integer size being targeted.

ACTUAL vs. EXPECTED RESPONSE

A. Actual Response

When the required conditions are encountered, the processor provides the following response:

- Return the MAXNEG value (8000h for FIST16 & 80000000h for FIST32) to memory.
- The IE (Invalid Operation) bit in the Floating-Point status word is *not set* to flag the use of an invalid operand.
- The PE (precision error) bit in the Floating-Point status word is *set*.
- No exception handler is invoked.
- In the case of a FISTP instruction the Operand will have been popped from the floating-point stack.

B. Expected Response

The expected processor response when the invalid operation exception is *masked* is:

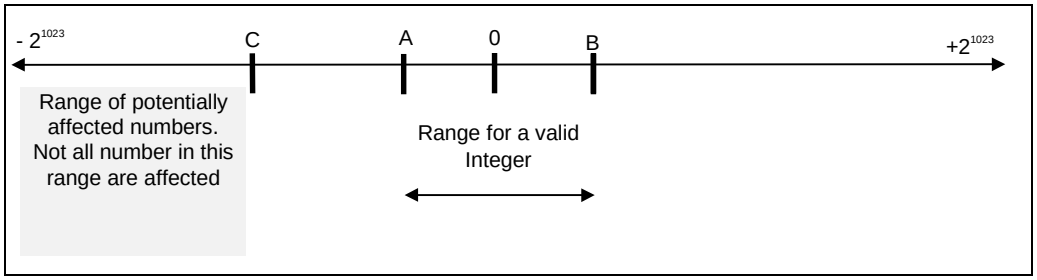
- Return the MAXNEG value (8000h for FIST16 & 80000000h for FIST32) to memory.
- The IE (Invalid Operation) bit in the Floating-Point status word is *set* to flag the overflow.
- The PE (precision error) bit in the Floating-Point status word is *not set*.

The expected processor response when the invalid operation exception is *unmasked* is:

- Do not return a result to memory. Keep the original operand intact on the stack.
- The IE (Invalid Operation) bit in the Floating-Point status word is *set* to flag the overflow.
- The PE (precision error) bit in the Floating-Point status word is *not set*.

Vector to the user numeric exception handler.

Implication: Erroneous operation results when the operand is so large that it will not fit into the target data size. The operands affected by this erratum are significantly outside (by a factor of 3X) the range that can be, correctly, converted to an integer value. The figure below and corresponding table identifies the normal range of integer numbers (between A and B) and the starting point of the operands affected by this erratum. Discrete failing operands will be present in the range between point C and the maximum negative number that can be represented by the processor (-2^{1023} in double precision format). Note 2 below gives a qualitative description of the nature of the discrete failing values. Software that does not rely on the Invalid Operation exception flag being set and signaled by either an exception OR by software polling is not impacted by this erratum.



16-bit Operation	A	B	C
	-32,768.0	+32,767.0	< -98304.0
32-bit Operation	A	B	C
	-2,147,483,648.0	+2,147,483,647.0	< -6,442,450,944.0

Workaround: Any of two software workarounds will avoid occurrence of this erratum:

1. Range checking performed prior to execution of the FIST[P] instruction will prevent the overflow condition from occurring, and may already be implemented as a coding style.
2. Software can use the presence of MAXNEG in the result integer to indicate that an out of range conversion may have occurred.

Note1: A possible alternative is to use the FIST64 instruction to store the converted operand to memory and access the lower 16 or 32 bits as the required integer. Even though this mechanism will not signal an attempted out of range conversion with a 16 bit or 32 bit target, it is currently in use by many compilers today.

Note2: The values affected by this erratum are those which contain an exponent value within the affected range, AND a specific bit pattern at a specific offset within the mantissa, AND at least one nonzero bit to the right of the above bit pattern. The offset within the mantissa is a function of the floating-point exponent value. The specific bit pattern is 0x8000 for FIST16 and 0x80000000 for FIST32. This means that for any given exponent within the range, one mantissa value in every 2^{16} possible mantissa values exhibits the erratum for FIST16, and one mantissa value in every 2^{32} possible mantissa values exhibits the erratum for FIST32.

Examples of affected values for FIST16, 80-bit binary notation (not an exhaustive list)
(xx means any bit pattern, yy means any nonzero bit pattern)

sign	exponent	mantissa
1	100000000010100	1xxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000010101	1xxxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000010110	1xxxxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

Examples of affected values for FIST32, 80-bit binary notation (not an exhaustive list)
(xx means any bit pattern, yy means any nonzero bit pattern)

sign	exponent	mantissa
1	100000000100011	1xxxx10000000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000100100	1xxxxx10000000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000100101	1xxxxxx10000000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A26. FLUSH# Assertion Disables L2 Machine Check Exception Reporting

Problem: Upon FLUSH# assertion, the L2 Machine Check Exception generation is disabled. Once the FLUSH# pin is asserted, the processor disables the L2 MCA, by clearing the associated MCi_CTL control register to “0”s. This operation is invisible to the software being executed.

Implication: Errors that should be reported by the L2 MCA are not reported from the time that the FLUSH# signal is asserted until the time that the MCi_CTL register is written back to all “1”s. All other errors will continue to be logged as normal.

Workaround: Platform specific code (e.g., BIOS or system management software) has the potential for driving a device to assert the FLUSH# pin. If the platform specific code asserts the FLUSH# pin, this code should be enhanced to detect that MCA Exceptions are globally enabled (via register CR4.MCE). The code should then write “0”s to all of the MCi_CTL registers to clear any spurious entries and then write “1”s to all of the MCi_CTL registers in order to re-enable exception reporting. Hardware devices in systems that require L2 error reporting which could assert the FLUSH# pin should not assert FLUSH#.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A27. EFLAGS May Be Incorrect After a Multiprocessor TLB Shootdown

Problem: When the Pentium II processor executes a read-modify-write arithmetic instruction with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes. In this case the EFLAGS value pushed onto the stack of the page fault handler may be reflective of the status of the EFLAGS register after the instruction would have completed execution rather than that before it has executed under a certain set of circumstances. This class of instruction will initially perform a load operation that has the side effect of ensuring that the final store portion of the instruction will successfully complete. The load ensures this by bringing the page table information of the page containing the data into the DTLB. This page entry could be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB, before the store is executed. DTLB eviction will require at least three load operations that have linear address bits 15:12 equal to each other and address bits 31:16 different from each other in close physical proximity to the arithmetic operation. If, in the very small window of time between the page eviction and the store execution, the page table entry has had its page permissions tightened (e.g., from Present to Not Present, or from Read/Write to Read Only, etc.) by the operating system in main memory by another processor (with no corresponding synchronization and subsequent TLB flush), the store will generate a DTLB miss and a call to the OS's page fault handler. The EFLAGS register may have already been updated by the arithmetic portion of the instruction before entry to the page fault handler. If under these circumstances the fault handler elects to restart the instruction, the re-execution may generate an incorrect result. Instructions affected by this erratum are the memory destination forms of ADC, SBB, RCR & RCL (instructions that use a flag, carry, as input to the instruction). It should be noted that the locked version of these instructions is not impacted by this erratum.

Implication: This scenario can only occur in a multiprocessor system running under an operating system that implements a "lazy" TLB shootdown. Lazy TLB shootdown occurs when one processor makes changes to the page tables in memory, and then signals other processors to remove the page entry from their TLB without a multiprocessor synchronization being performed. To date, Intel has not observed this erratum in any laboratory testing of commercially available software applications.

In a multiprocessor system the arithmetic flags of the EFLAGS register and its memory stack image, may contain incorrect data if the read-modify-write arithmetic instruction encounters a page fault. Page Fault handler software that uses the resulting EFLAGS may see incorrect information. If the original instruction is restarted by the page fault handler, the instruction may produce incorrect results based on the prior modifications of the EFLAGS register.

Workaround: Software may use the locked form of the ADC, SBB, RCR & RCL instructions to avoid this erratum. Operating systems should ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened, e.g., moved from Present to Not Present or have a page fault handler that can handle any incorrect state. Intel is working with Multiprocessor Operating System vendors to ensure that an OS level workaround is implemented as required.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A28. Delayed Line Invalidation Issue During 2-Way MP Data Ownership Transfer

Problem: In 2-way MP systems, each processor may attempt to modify a different portion of the same cache line, referenced as line 'A' in the discussion below. When this erratum occurs (with the following example given for a 2-way MP system with processors noted as 'P0' and 'P1'), each processor contains a shared copy of line A in both their L1 and L2 caches. Each processor must issue an invalidation cycle before that processor can definitively source the results of its internal write to a portion of line A to the other processors.

There exists a narrow timing window when, if P0 wins the external bus invalidation race and gains ownership rights to line A due to the sequence of bus invalidation traffic, P1 may not have completed the pending invalidation of its own, currently valid and shared copy of line A. During this window, it is possible for a P1 internal opportunistic write to a portion of line A (while awaiting ownership rights) to occur with the original shared copy of line A still resident in P1's L2 cache. Such internal modification is permissible subject to delaying the broadcast of such changes until line ownership has actually been gained. However, the processor must ensure that any internal re-read by P1 of line A returns with data in the order actually written; in this case, this should be the data written by P0. In the case of this erratum, the internal re-read uses the data which was written by P1.

Implication: Multiprocessor or threaded application synchronization that is implemented via operating system-provided synchronization constructs are not affected by this erratum. Applications which rely upon the usage of locked semaphores rather than memory ordering are also unaffected. Uniprocessor systems are not affected by this erratum. Intel has not identified, to date, any commercially available application or operating system software which is affected by this erratum. If the erratum does occur, the delayed line invalidation that occurs naturally due to the fact that one processor will necessarily win the invalidation race allows a narrow timing window to exist where one processor may re-read a line that it just wrote internally, but return with the stale data that was present from the previous shared state rather than the data written more recently by another processor.

Workaround: Deterministic barriers beyond which program variables will not be modified can be achieved via the usage of locked semaphore operations, and this scheme has been shown to effectively work around this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A29. Potential Early Deassertion of LOCK# During Split-Lock Cycles

Problem: During a split-lock cycle there are four bus transactions: 1st ADS# (a partial read), 2nd ADS# (a partial read), 3rd ADS# (a partial write), and the 4th ADS# (a partial write). Due to this erratum, LOCK# may deassert one clock after the 4th ADS# of the split-lock cycle instead of after the 4th RS# assertion corresponding to the 4th ADS# has been sampled. The following sequence of events are required for this erratum to occur:

1. A lock cycle occurs (split or nonsplit).
2. Five more bus transactions (assertion of ADS#) occur.
3. A split-lock cycle occurs and BNR# toggles after the 3rd ADS# (partial write) of the split-lock cycle. This in turn delays the assertion of the 4th ADS# of the split-lock cycle. BNR# toggling at this time could most likely happen when the bus is set for an IOQ depth of 2.

When all of these events occur, LOCK# will be deasserted in the next clock after the 4th ADS# of the split-lock cycle.

Implication: This may affect chipset logic which monitors the behavior of LOCK# deassertion.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A30. A20M# May Be Inverted After Returning From SMM and Reset

Problem: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of nonvolatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.
3. After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

Implication: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (i.e., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).

Workaround: Software should save the A20M# signal state in nonvolatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A31. Reporting of Floating-Point Exception May Be Delayed

Problem: The Pentium II processor normally reports a floating-point exception for an instruction when the next floating-point or MMX™ technology instruction is executed. The assertion of FERR# and/or the INT 16 interrupt corresponding to the exception may be delayed until the floating-point or MMX technology instruction *after* the one which is expected to trigger the exception, if the following conditions are met:

1. A floating-point instruction causes an exception.
2. Before another floating-point or MMX technology instruction, any one of the following occurs:
   a. A subsequent data access occurs to a page which has not been marked as accessed, or
   b. Data is referenced which crosses a page boundary, or
   c. A possible page-fault condition is detected which, when resolved, completes without faulting.
3. The instruction causing event 2 above is followed by a MOVQ or MOVD store instruction.

Implication: This erratum only affects software which operates with floating-point exceptions unmasked. Software which requires floating-point exceptions to be visible on the next floating-point or MMX technology instruction, and which uses floating-point calculations on data which is then used for MMX technology instructions, may see a delay in the reporting of a floating-point instruction exception in some cases. Note that mixing floating-point and MMX technology instructions in this way is not recommended.

Workaround: Inserting a WAIT or FWAIT instruction (or reading the floating-point status register) between the floating-point instruction and the MOVQ or MOVD instruction will give the expected results. This is already the recommended practice for software.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A32. EFLAGS Discrepancy on a Page Fault After a Multiprocessor TLB Shootdown

Problem: This erratum may occur when the Pentium II processor executes one of the following read-modify-write arithmetic instructions and a page fault occurs during the store of the memory operand: ADD, AND, BTC, BTR, BTS, CMPXCHG, DEC, INC, NEG, NOT, OR, ROL/ROR, SAL/SAR/SHL/SHR, SHLD, SHRD, SUB, XOR, and XADD. In this case, the EFLAGS value pushed onto the stack of the page fault handler may reflect the status of the register after the instruction would have completed execution rather than before it. The following conditions are required for the store to generate a page fault and call the operating system page fault handler:

1. The store address entry must be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB before the store has completed. DTLB eviction requires at least three load operations that have linear address bits 15:12 equal to each other and address bits 31:16 different from each other in close physical proximity to the arithmetic operation.
2. The page table entry for the store address must have its permissions tightened during the very small window of time between the DTLB eviction and execution of the store. Examples of page permission tightening include from Present to Not Present or from Read/Write to Read Only, etc.
3. Another processor, without corresponding synchronization and TLB flush, must cause the permission change.

Implication: This scenario may only occur on a multiprocessor platform running an operating system that performs “lazy” TLB shootdowns. The memory image of the EFLAGS register on the page fault handler’s stack prematurely contains the final arithmetic flag values although the instruction has not yet completed. Intel has not identified any operating systems that inspect the arithmetic portion of the EFLAGS register during a page fault nor observed this erratum in laboratory testing of software applications.

Workaround: No workaround is needed upon normal restart of the instruction, since this erratum is transparent to the faulting code and results in correct instruction behavior. Operating systems may ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened or have a page fault handler that ignores any incorrect state.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A33. Near CALL to ESP Creates Unexpected EIP Address

Problem: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP *after* the return value is pushed on the stack, not the value in the ESP register *before* the instruction executed.

Implication: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

Workaround: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an *indirect* call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A34. Deep Sleep Exit Transition May Cause Hang

Problem: Under normal operating conditions, when a system enters a power conservation mode, it enters System Management Mode (SMM), puts the processor in the Stop Grant State, followed by Sleep State and then may enter Deep Sleep State. Upon a resume event, the processor exits Deep Sleep but remains in SMM execution space until the SMI handler completes the system resume cycle.

If, prior to entering the Deep Sleep, the system was in SMM space, it is possible for the processor to exit Deep Sleep state and begin making accesses in the 'normal' memory space instead of staying in SMM space. The converse is also possible, i.e., if the processor is in 'normal' space prior to entering the Deep Sleep state, the processor may exit Deep Sleep and make accesses in SMM space instead.

Implication: Systems may execute incorrect code after exiting Deep Sleep, due to accesses to incorrect address space. This may produce unpredictable behavior, most likely hanging the system.

Workaround: Avoid entering Deep Sleep. The table below offers the possible state transitions:

#	System State		Processor State		Possible Solutions	
	Name	ACPI Equivalent	Transition	ACPI Equivalent	Description	Suggested Solution
1	Active	S0	Normal to Stop Grant	C0, C1	N/A	None necessary

#	System State		Processor State		Possible Solutions	
	Name	ACPI Equivalent	Transition	ACPI Equivalent	Description	Suggested Solution
2	Active	S0	Stop Grant to Sleep to Deep Sleep	C1, C3	Use Stop Grant/Sleep only; do not use Deep Sleep	BIOS can specify the C3 latency time to be >1000 •s in the ACPI FACP table (P_LVL3_LAT, worst case hardware latency for the C3 state).
3	Powered On Suspend	S1, S2	Stop Grant to Deep Sleep	C1, C3	Reset CPU only and flush the cache without resetting the PCI bus, i.e., use POS_CCL state (POS with CPU Context Lost) instead of POS state.	BIOS can prevent the OS from entering the S1 state by NOT defining the S1 object in the ACPI DSDT table. Ensure that the cache is always flushed.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A35. Built-in Self Test Always Gives Nonzero Result

Problem: The Built-in Self Test (BIST) of the Pentium II processor does not give a zero result to indicate a passing test. Regardless of pass or fail status, bit 6 of the BIST result in the EAX register after running BIST is set.

Implication: Software which relies on a zero result to indicate a passing BIST will indicate BIST failure.

Workaround: Mask bit 6 of the BIST result register when analyzing BIST results.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A36. THERMTRIP# May Not Be Asserted as Specified

Problem: THERMTRIP# is a signal on the Pentium II processor which is asserted when the core reaches a critical temperature during operation as detailed in the processor specification. The Pentium II processor may not assert THERMTRIP# until a much higher temperature than the one specified is reached.

Implication: The THERMTRIP# feature is not functional on the Pentium II processor. Note that this erratum can only occur when the processor is running with a T_{PLATE} temperature over the maximum specification of 75° C.

Workaround: Avoid operation of the Pentium II processor outside of the thermal specifications defined by the processor specifications.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A37. Cache State Corruption in the Presence of Page A/D-bit Setting and Snoop Traffic

Problem: If an operating system uses the Page Access and/or Dirty bit feature implemented in the Intel architecture and there is a significant amount of snoop traffic on the bus, while the processor is setting the Access and/or Dirty bit the processor may inappropriately change a single L1 cache line to the modified state.

Implication: The occurrence of this erratum may result in cache incoherency, which may cause parity errors, data corruption (with no parity error), unexpected application or operating system termination, or system hangs.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A38. Snoop Cycle Generates Spurious Machine Check Exception

Problem: The processor may incorrectly generate a Machine Check Exception (MCE) when it processes a snoop access that does not hit the L1 data cache. Due to an internal logic error, this type of snoop cycle may still check data parity on undriven data lines. The processor generates a spurious machine check exception as a result of this unnecessary parity check.

Implication: A spurious machine check exception may result in an unexpected system halt if Machine Check Exception reporting is enabled in the operating system.

Workaround: It is possible for BIOS code to contain a workaround for this erratum. This workaround would fix the erratum, however, the data parity error will still be reported.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A39. MOVD/MOVQ Instruction Writes to Memory Prematurely

Problem: When an instruction encounters a fault, the faulting instruction should not modify any CPU or system state. However, when the MMX™ technology store instructions MOVD and MOVQ encounter any of the following events, it is possible for the store to be committed to memory even though it should be canceled:

1. If CR0.EM = 1 (Emulation bit), then the store could happen prior to the triggered invalid opcode exception.
2. If the floating-point Top-of-Stack (FP TOS) is not zero, then the store could happen prior to executing the processor assist routine that sets the FP TOS to zero.
3. If there is an unmasked floating-point exception pending, then the store could happen prior to the triggered unmasked floating-point exception.
4. If CR0.TS = 1 (Task Switched bit), then the store could happen prior to the triggered Device Not Available (DNA) exception.

If the MOVD/MOVQ instruction is restarted after handling any of the above events, then the store will be performed again, overwriting with the expected data. The instruction will not be restarted after event 1. The instruction will definitely be restarted after events 2 and 4. The instruction may or may not be restarted after event 3, depending on the specific exception handler.

Implication: This erratum causes unpredictable behavior in an application if MOVD/MOVQ instructions are used to manipulate semaphores for multiprocessor synchronization, or if these MMX instructions are used to write to uncacheable memory or memory mapped I/O that has side effects, e.g., graphics devices. This erratum is completely transparent to all applications that do not have these characteristics. When each of the above conditions are analyzed:

1. Setting the CR0.EM bit forces all floating-point/MMX instructions to be handled by software emulation. The MOVD/MOVQ instruction, which is an MMX instruction, would be considered an invalid instruction. Operating systems typically terminates the application after getting the expected invalid opcode fault.
2. The FP TOS not equal to 0 case only occurs when the MOVD/MOVQ store is the first MMX instruction in an MMX technology routine and the previous floating-point routine did not clean up the floating-point states properly when it exited. Floating-point routines commonly leave TOS to 0 prior to exiting. For a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.
3. The unmasked floating-point exception case only occurs if the store is the first MMX technology instruction in an MMX technology routine and the previous floating-point routine exited with an unmasked floating-point exception pending. Again, for a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.

Device Not Available (DNA) exceptions occur naturally when a task switch is made between two tasks that use either floating-point instructions and/or MMX instructions. For this erratum, in the event of the DNA exception, data from the prior task may be temporarily stored to the present task's program state.

Workaround: Do not use MMX instructions to manipulate semaphores for multiprocessor synchronization. Do not use MOVD/MOVQ instructions to write directly to I/O devices if doing so triggers user visible side effects. An OS can prevent old data from being stored to a new task's program state by cleansing the FPU explicitly after every task switch. Follow Intel's recommended programming paradigms in the *Intel Architecture Optimization Manual* for writing MMX technology programs. Specifically, do not mix floating-point and MMX instructions. When transitioning to new a MMX technology routine, begin with an instruction that does not depend on the prior state of either the MMX technology registers or the floating-point registers, such as a load or PXOR mm0, mm0. Be sure that the FP TOS is clear before using MMX instructions.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A40. Memory Type Undefined for Nonmemory Operations

Problem: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.

Implication: Bus agents may decode a non-UC memory type for nonmemory bus transactions.

Workaround: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A41. Infinite Snoop Stall During L2 Initialization of MP Systems

Problem: It is possible for snoop traffic generated on the system bus while a processor is executing its L2 cache initialization routine to cause the initializing processor to hang.

Implication: A DP (2-way) system which does not suppress snoop traffic while L2 caches are being initialized may hang during this initialization sequence.

Workaround: System BIOS can create an execution environment which allows processors to initialize their L2 caches without the system generating any snoop traffic on the bus.

Below is a pseudo-code fragment, designed explicitly for a two-processor system, that uses a serial algorithm to initialize each processor's L2 cache:

```
Suppress_all_I/O_traffic()
 K = 0;
 while (K <= 1)
 {
 /* Obtain current value of K. This forces both Temp and K into */
 /* the L1 cache. Note that Temp could also be maintained in a */
 /* general purpose register. */

 Temp = K;
 Wait_until_all_processors_are_signed_in_at_barrier()
 if ( logical_proc_APIC_id == K ) {
 {
 wait_10_usecs_delay_loop(); /* this time delay, required */
 /* in the worst case, allows */
 /* the barrier semaphore to */
 /* settle to shared state. */
 Initialize L2 cache
 K++
 }
 else
 while (Temp == K);
 }
 }
```

This algorithm prevents bus snoop traffic from the other processors, which would otherwise cause the initializing processor to hang. The algorithm assumes that the L1 cache is enabled (the Temp and K variables must be cached by each processor). Also, the Memory Type Range Register (MTRR) for the data segment must be set to WB (writeback) memory type.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A42. Bus Protocol Conflict With Optimized Chipsets

Problem: A “dead” turnaround cycle with no agent driving the address, address parity, request command, or request parity signals must occur between the processor driving these signals and the chipset driving them after asserting BPRI#. The Pentium II processor does not follow this protocol. Thus, if a system uses a chipset or third party agent which optimizes its arbitration latency (reducing it to 2 clocks when it observes an active (low) ADS# signal and an inactive (high) LOCK# signal on the same clock that BPRI# is asserted (driven low)), the Pentium II processor may cause bus contention during an unlocked bus exchange.

Implication: This violation of the bus exchange protocol when using a reduced arbitration latency may cause a system-level setup timing violation on the address, address parity, request command, or request parity signals on the system bus. This may result in a system hang or assertion of the AERR# signal, causing an attempted corrective action or shutdown of the system, as the system hardware and software dictate. The possibility of failure due to the contention caused by this erratum may be increased due to the processor’s internal active pull-up of these signals on the clock after the signals are no longer being driven by the processor.

Workaround: Ensure that OS code does not clear the D-bit for system pages (including any pages that contain a task gate or TSS). Use task gates rather than jumping to a new TSS when performing a task switch.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A43. FP Data Operand Pointer May Not Be Zero After Power On or Reset

Problem: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

Implication: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting in incorrect behavior of the software.

Workaround: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer’s Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A44. MOVD Following Zeroing Instruction Can Cause Incorrect Result

Problem: An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits; or a signed integer multiply is performed to the same register's lower 16 bits,
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX technology register. Only the MMX technology register is affected by this erratum.

The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
   or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
   or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
   or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
   or IMUL AX, BX, 1024 (opcode 69 /r iw)
   or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw) or IMUL AX, 1024 (opcode 69 /r iw)
   or CBW
3. MOVD MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size may be affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX or IMUL instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

Implication: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

Workaround: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD, IMUL-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX technology for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX
```

*Note: MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A45. Premature Execution of a Load Operation Prior to Exception Handler Invocation

Problem: This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation,
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending, or
3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit set), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending.

If any of the above circumstances occur it is possible that the load portion of the instruction will have executed before the exception handler is entered.

Implication: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side-effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side-effect.

Workaround: Code which performs loads from memory that has side-effects can effectively workaround this behavior by using simple integer-based load instructions when accessing side-effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

Status: For the steppings affected see the Summary Table of Changes at the beginning of this section.

A46. Read Portion of RMW Instruction May Execute Twice

Problem: When the Pentium II processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.

If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

Implication: This erratum has no effect if the memory targeted for the RMW instruction has no side-effects. If, however, the load targets a memory region that has side-effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

Workaround: Hardware and software developers who write device drivers for custom hardware that may have a side-effect style of design should use simple loads and simple stores to transfer data to and from the device. Then, the memory location will simply be read twice with no additional implications.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A47. Test Pin Must Be High During Power Up

Problem: The processor uses the PWRGOOD signal to ensure that no voltage sequencing issues arise; no pin assertions should cause the processor to change its behavior until this signal is asserted, when all power supplies and clocks to the processor are valid and stable. However, if the TESTHI signal is at a low voltage level when the core power supply comes up, it will cause the processor to enter an invalid test state.

Implication: If this erratum occurs, the system may boot normally however, L2 cache may not be initialized.

Workaround: Ensure that the 2.5 V ($V_{CC2.5}$) power supply ramps at or before the 2.0 V ($V_{CC_{CORE}}$) power plane. If 2.5 V ramps after core, pull up TESTHI to 2.5 V ($V_{CC2.5}$) with a 100K ohm resistor. The internal pull-up will keep the signal from being asserted during power up. For new motherboard designs, it is recommended that TESTHI be pulled up to 2.0 V ($V_{CC_{CORE}}$) using a 1K-10K ohm resistor.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A48. Intervening Writeback May Occur During Locked Transaction

Problem: During a transaction which has the LOCK# signal asserted (i.e., a locked transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations will occur. This erratum is, however, a violation of the bus lock protocol.

Implication: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that locked transactions may only consist of a read-write or read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.

Workaround: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a locked transaction.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A49. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

Problem: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field, and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

Implication: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

Workaround: When decoding the MC2_STATUS MSR, reverse the two error fields.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A50. Mixed Cacheability of Lock Variables Is Problematic in MP Systems

Problem: This errata only affects multiprocessor systems where a lock variable address is marked cacheable in one processor and uncacheable in any others. The processors which have it marked uncacheable may stall indefinitely when accessing the lock variable. The stall is only encountered if:

- One processor has the lock variable cached, and is attempting to execute a cache lock.
- If the processor which has that address cached has it cached in its L2 only.

Other processors, meanwhile, issue back to back accesses to that same address on the bus.

Implication: MP systems where all processors either use cache locks or consistent locks to uncacheable space will not encounter this problem. If, however, a lock variable's cacheability varies in different processors, and several processors are all attempting to perform the lock simultaneously, an indefinite stall may be experienced by the processors which have it marked uncacheable in locking the variable (if the conditions above are satisfied). Intel has only encountered this problem in focus testing with artificially generated external events. Intel has not currently identified any commercial software which exhibits this problem.

Workaround: Follow a homogenous model for the memory type range registers (MTRRs), ensuring that all processors have the same cacheability attributes for each region of memory; do not use locks whose memory type is cacheable on one processor, and uncacheable on others. Avoid page table aliasing, which may produce a nonhomogenous memory model.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A51. MOV With Debug Register Causes Debug Exception

Problem: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 15.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

Implication: With debug-register protection enabled (i.e., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

Workaround: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A52. Upper Four PAT Entries Not Usable With Mode B or Mode C Paging

Problem: The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the Pentium II processor. However, in Mode B or Mode C paging, the upper four entries do not function correctly for 4-Kbyte pages. Specifically, bit seven of page table entries that translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE) are enabled, the processor forces this bit to zero when determining the memory type regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).

Implication: Only the lower four PAT entries are useful for 4 KB translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may be used for large pages in Mode B or C paging.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A53. UC Write May Be Reordered Around a Cacheable Write

Problem: After a write occurs to a UC (uncacheable) region of memory, there exists a small window of opportunity where a subsequent write transaction targeted for a UC memory region may be reordered in front of a write targeted to a region of cacheable memory. This erratum can only occur during the following sequence of bus transactions:

- A write to memory mapped as UC occurs,
- A write to memory mapped as cacheable (WB or WT) which is present in Shared or Invalid state in the L2 cache occurs, and
- During the bus snoop of the cacheable line, another store to UC memory occurs.

Implication: If this erratum occurs, the second UC write will be observed on the bus prior to the Bus Invalidate Line (BIL) or Bus Read Invalidate Line (BRIL) transaction for the cacheable write. This presents a small window of opportunity for a fast bus-mastering I/O device which triggers an action based on the second UC write to arbitrate and gain ownership of the bus prior to the completion of the cacheable write, possibly retrieving stale data.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A54. Incorrect Memory Type May Be Used When MTRRs Are Disabled

Problem: If the Memory Type Range Registers (MTRRs) are disabled without setting the CR0.CD bit to disable caching, and the Page Attribute Table (PAT) entries are left in their default setting, which includes UC- memory type (PCD = 1, PWT = 0; see the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* for details), data for entries set to UC- will be cached as if the memory type were writeback (WB). Also, if the page tables are set to a memory type other than UC-, then the effective memory type used will be that specified by the page tables and PAT. Any regions of memory normally forced to UC by the MTRRs (such as the VGA video region) may now be incorrectly cached and speculatively accessed.

Even if the CR0.CD bit is correctly set when the MTRRs are disabled and the PAT is left in its default state, then retries and out of order retirement of UC accesses may occur, contrary to the strong ordering expected for these transactions.

Implication: The occurrence of this erratum may result in the use of incorrect data and unpredictable processor behavior when running with the MTRRs disabled. Interaction between the mouse, cursor, and VGA video display leading to video corruption may occur as a symptom of this erratum as well.

Workaround: Ensure that when the MTRRs are disabled, the CR0.CD bit is set to disable caching. This recommendation is described in *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. If it is necessary to disable the MTRRs, first clear the PAT register before setting the CR0.CD bit, flushing the caches, and disabling the MTRRs to ensure that UC memory type is always returned and strong ordering is maintained.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A55. Misprediction in Program Flow May Cause Unexpected Instruction Execution

Problem: To optimize performance through dynamic execution technology, the P6 architecture has the ability to predict program flow. In the event of a misprediction, the processor will normally clear the incorrect prediction, adjust the EIP to the correct location, and flush out any instructions it may have fetched from the misprediction. In circumstances where a branch misprediction occurs, the correct target of the branch has already been opportunistically fetched into the streaming buffers, and the L2 cycle caused by the evicted cache line is retried by the L2 cache, the processor may fail to flush out the retirement unit before the speculative program flow is committed to a permanent state.

Implication: The results of this erratum may range from no effect to unpredictable application or OS failure. Manifestations of this failure may result in:

- Unexpected values in EIP,
- Faults or traps (e.g., page faults) on instructions that do not normally cause faults,
- Faults in the middle of instructions, or
- Unexplained values in registers/memory at the correct EIP.

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A56. System Bus ECC May Report False Errors

Problem: The processor's ECC circuitry may fail to meet its frequency timing specification under certain environmental conditions. At the high end of the temperature specification and/or the low end of the voltage range, the processor may report false ECC errors.

Implication: If the system has data error checking enabled (bit [1] of the EBL_CR_POWERON register set to "1") and has Machine Check Architecture enabled, spurious double bit error detection can occur causing Machine Check Exceptions (MCE) and spurious single bit errors to occur and be logged. Under some circumstances the processor may assert BINIT#, which in turn, may cause some systems to generate an MCE, and in others cause a reboot.

Workaround: Disable system bus data error checking (set bit [1] of the EBL_CR_POWERON register to "0").

Status: For the processor part numbers affected see the "Pentium® II Processor Identification Information" table in the *General Information* section.

A57. Full In-Order Queue May Cause Infinite DBSY# Assertion

Problem: For this erratum to occur, there must be a high rate of code fetches from the core to its L2 cache, which must hit the L2 cache, AND in parallel an externally generated read transaction that hits a modified line FOLLOWED by 7 consecutive 0 length external transactions in rapid succession FOLLOWED by another external transaction that also hits a modified line.

Implication: The writeback data is not transferred to memory. No further bus transactions may be issued because the In-Order Queue is full.

Workaround: None Identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A58. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

Problem: If a misaligned data breakpoint is programmed to the same cache line as the memory location where the stack push of a near call is performed and any data breakpoints are enabled, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.

Implication: The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.

Workaround: Whether enabled or not, do not program a misaligned data breakpoint to the same cache line on the stack where the push for the near call is performed.

Status: For the stepping affected see the *Summary of Changes* at the beginning of this section.

A59. System Bus ECC Not Functional With 2:1 Ratio

Problem: If a processor is underclocked at a core frequency to system bus frequency ratio of 2:1 and system bus ECC is enabled, the system bus ECC detection and correction will negatively affect internal timing dependencies.

Implication: If system bus ECC is enabled, and the processor is underclocked at a 2:1 ratio, the system may behave unpredictably due to these timing dependencies.

Workaround: All bus agents that support system bus ECC must disable it when a 2:1 ratio is used.

Status: For the steppings affected see the *Summary of Change*s at the beginning of this section.

A60. Fault on REP CMPS/SCAS Operation May Cause Incorrect EIP

Problem: If either a General Protection Fault, Alignment Check Fault or Machine Check Exception occur during the first iteration of a REP CMPS or a REP SCAS instruction, an incorrect EIP may be pushed onto the stack of the event handler if all the following conditions are true:

- The event occurs on the initial load performed by the instruction(s),
- The condition of the zero flag before the repeat instruction happens to be opposite of the repeat condition (i.e., REP/REPE/REPZ CMPS/SCAS with ZF = 0 or RENE/REPNZ CMPS/SCAS with ZF = 1), and
- The faulting micro-op and a particular micro-op of the REP instruction are retired in the retirement unit in a specific sequence.

The EIP will point to the instruction following the REP CMPS/SCAS instead of pointing to the faulting instruction.

Implication: The result of the incorrect EIP may range from no effect to unexpected application/OS behavior.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A61. RDMSR or WRMSR To Invalid MSR Address May Not Cause GP Fault

Problem: The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

Implication: For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

Workaround: Do not use invalid MSR addresses with RDMSR or WRMSR.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A62. SYSENTER/SYSEXIT Instructions Can Implicitly Load "Null Segment Selector" to SS and CS Registers

Problem: According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh, inclusive.

Implication: These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor's behavior may become unpredictable, possibly resulting in system failure.

Workaround: Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A63. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

Problem: According to the IEEE 1149.1 Standard, the EXTEST instruction would use data "typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction." As a result of this erratum, this method cannot be used to load the data onto the outputs.

Implication: Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

Workaround: None identified

Status: For the steppings affected see the *Summary of Change*s at the beginning of this section.

A64. Far Jump to New TSS With D-bit Cleared May Cause System Hang

Problem: A task switch may be performed by executing a far jump through a task gate or to a new Task State Segment (TSS) directly. Normally, when such a jump to a new TSS occurs, the D-bit (which indicates that the page referenced by a Page Table Entry (PTE) has been modified) for the PTE which maps the location of the previous TSS will already be set, and the processor will operate as expected. However, if the D-bit is clear at the time of the jump to the new TSS, the processor will hang.

Implication: If an OS is used which can clear the D-bit for system pages, and which jumps to a new TSS on a task switch, then a condition may occur which results in a system hang. Intel has not identified any commercial software which may encounter this condition; this erratum was discovered in a focused testing environment.

Workaround: Ensure that OS code does not clear the D-bit for system pages (including any pages that contain a task gate or TSS). Use task gates rather than jumping to a new TSS when performing a task switch.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A65. Incorrect Chunk Ordering May Prevent Execution of the Machine Check Exception Handler After BINIT#

Problem: If a catastrophic bus error is detected which results in a BINIT# assertion, and the BINIT# assertion is propagated to the processor's L2 cache at the same time that data is being sent to the processor, then the data may become corrupted in the processor's L1 cache.

Implication: Since BINIT# assertion is due to a catastrophic event on the bus, the corrupted data will not be used. However, it may prevent the processor from executing the Machine Check Exception (MCE) handler, causing the system to hang.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A66. Resume Flag May Not Be Cleared After Debug Exception

Problem: The Resume Flag (RF) is normally cleared by the processor after executing an instruction which causes a debug exception (#DB). In the process of determining whether the RF needs to be cleared after executing the instruction, the processor uses an internal register containing stale data. The stale data may unpredictably prevent the processor from clearing the RF.

Implication: If this erratum occurs, further debug exceptions will be disabled.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A67. System Bus Address Parity Generator May Report False AERR#s

Problem: The processor's address parity error detection circuit may fail to meet its frequency timing specification under certain environmental conditions. At the high end of the temperature specification and/or the low end of the voltage range, the processor may report false address parity errors (AERRs).

Implication: If the system has AERR# drive enabled (bit [3] of the EBL_CR_POWERON resister set to '1') spurious address detection and reporting may occur. In some system configurations BINIT# may be asserted on the system bus. This may cause some systems to generate a machine check exception and in others may cause a reboot.

Workaround: Disable AERR# drive from the processor. AERR# drive may be disabled by clearing bit [3] in the EBL_CR_POWERON register. In addition, if the chipset allows, AERR# drive should be enabled from the chipset and AERR# observation enabled on the processor. AERR# observation on the processor is enabled by asserting A8# on the active-to-inactive transition of RESET#.

Status: For the processor part numbers affected see the "Pentium® II Processor Identification and Packaging Information" table at the General Information section.

A68. Misaligned Locked Access to APIC Space Results in Hang

Problem: When the processor's APIC space is accessed with a misaligned locked access a machine check exception is expected. However, the processor's machine check architecture is unable to handle the misaligned locked access.

Implication: If this erratum occurs the processor will hang. Typical usage models for the APIC address space do not use locked accesses. This erratum will not affect systems using such a model.

Workaround: Ensure that all accesses to APIC space are aligned.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A69. Potential Loss of Data Coherency During MP Data Ownership Transfer

Problem: In MP systems, processors may be sharing data in different cache lines, referenced as line A and line B in the discussion below. When this erratum occurs (with the following example given for a 2-way MP system with processors noted as ‘P0’ and ‘P1’), P0 contains a shared copy of line B in its L1. P1 has a shared copy of Line A. Each processor must manage the necessary invalidation and snoop cycles before that processor can modify and source the results of any internal writes to the other processor.

There exists a narrow timing window when, if P1 requests a copy of line B it may be supplied by P0 in an exclusive state which allows P1 to modify the contents of the line with no further external invalidation cycles. In this narrow window P0 may also retire instructions that use the original data present before P1 performed the modification.

Implication: Multiprocessor or threaded application synchronization, required for low-level data sharing, that is implemented via operating system provided synchronization constructs are not affected by this erratum. Applications that rely upon the usage of locked semaphores rather than memory ordering are also unaffected. Uniprocessor systems are not affected by this erratum. If the erratum does occur, one processor may execute software with the stale data that was present from the previous shared state rather than the data written more recently by another processor.

Workaround: Deterministic barriers beyond which program variables will not be modified can be achieved via the usage of locked semaphore operations. These should effectively prevent the occurrence of this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A70. Memory Ordering Based Synchronization May Cause a Livelock Condition in MP Systems

Problem: In an MP environment, the following sequence of code (or similar code) in two processors (P0 and P1) may cause them to each enter an infinite loop (livelock condition):

```
        P0                                      P1
                MOV [xyz], EAX      (1)   wait1:  MOV EBX, [abc]    (2)
                .                                 CMP EBX, val1     (3)
                .                                 JNE wait1         (4)
                .
                MOV [abc], val1     (6)           MOV [abc], val2   (5)
wait0:  MOV EBX, [abc]      (7)
                CMP EBX, val2       (8)
                JNE wait0           (9)
```

NOTE

EAX and EBX can be any general-purpose register. Addresses [abc] and [xyz] can be any location in memory and must be in the same bank of the L1 cache. Variables “val1” and “val2” can be any integer.

The algorithm above involves processors P0 and P1, each of which use loops to keep them synchronized with each other. P1 is looping until instruction (6) in P0 is globally observed. Likewise, P0 will loop until instruction (5) in P1 is globally observed.

The P6 architecture allows for instructions (1) and (7) in P0 to be dispatched to the L1 cache simultaneously. If the two instructions are accessing the same memory bank in the L1 cache, the load (7) will be given higher priority and will complete, blocking instruction (1).

Instructions (8) and (9) may then execute and retire, placing the instruction pointer back to instruction (7). This is due to the condition at the end of the “wait0” loop being false. The livelock scenario can occur if the timing of the wait0 loop execution is such that instruction (7) in P0 is ready for completion every time that instruction (1) tries to complete. Instruction (7) will again have higher priority, preventing the data ([xyz]) in instruction (1) from being written to the L1 cache. This causes instruction (6) in P0 to not complete and the sequence “wait0” to loop infinitely in P0.

A livelock condition also occurs in P1 because instruction (6) in P0 does not complete (blocked by instruction (1) not completing). The problem with this scenario is that P0 should eventually allow for instruction (1) to write its data to the L1 cache. If this occurs, the data in instruction (6) will be written to memory, allowing the conditions in both loops to be true.

Implication: Both processors will be stuck in an infinite loop, leading to a hang condition. Note that if P0 receives any interrupt, the loop timing will be disrupted such that the livelock will be broken. The system timer, a keystroke, or mouse movement can provide an interrupt that will break the livelock.

Workaround: Use a LOCK instruction to force P0 to execute instruction (6) before instruction (7).

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A71. GP# Fault on WRMSR to ROB_CR_BKUPTMPDR6

Problem: Writing a '1' to unimplemented bit(s) in the ROB_CR_BKUPTMPDR6 MSR (offset 1E0h) will result in a general protection fault (GP#).

Implication: The normal process used to write an MSR is to read the MSR using RDMSR, modify the bit(s) of interest, and then to write the MSR using WRMSR. Because of this erratum, this process may result in a GP# fault when used to modify the ROB_CR_BKUPTMPDR6 MSR.

Workaround: When writing to ROB_CR_BKUPTMPDR6 all unimplemented bits must be '0.' Implemented bits may be set as '0' or '1' as desired.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A72. Machine Check Exception May Occur Due to Improper Line Eviction in the IFU

Problem: The Pentium II processor is designed to signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism. Under a complex set of circumstances involving multiple speculative branches and memory accesses there exists a one cycle long window in which the processor may signal a MCE in the Instruction Fetch Unit (IFU) because instructions previously decoded have been evicted from the IFU. The one cycle long window is opened when an opportunistic fetch receives a partial hit on a previously executed but not as yet completed store resident in the store buffer. The resulting partial hit erroneously causes the eviction of a line from the IFU at a time when the processor is expecting the line to still be present. If the MCE for this particular IFU event is disabled, execution will continue normally.

Implication: While this erratum may occur on a system with any number of Pentium II processors, the probability of occurrence increases with the number of processors. If this erratum does occur, a machine check exception will result. Note systems that implement an operating system that does not enable the Machine Check Architecture will be completely unaffected by this erratum (e.g., Windows* 95 and Windows 98).

Workaround: It is possible for BIOS code to contain a workaround for this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A73. Lower Bits of SMRAM SMBASE Register Cannot Be Written With an ITP

Problem: The System Management Base (SMBASE) register (7EF8H) stores the starting address of the System Management RAM (SMRAM). This register is used by the processor when it is in System Management Mode (SMM), and its contents serve as the memory base for code execution and data storage. The 32-bit SMBASE register can normally be programmed to any value. When programmed with an In-Target Probe (ITP), however, any attempt to set the lower 11 bits of SMBASE to anything other than zeros via the WRMSR instruction will cause the attempted write to fail.

Implication: When set via an ITP, any attempt to relocate SMRAM space must be made with 2 Kbyte alignment.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A74. Task Switch May Cause Wrong PTE and PDE Access Bit to be Set

Problem: If an operating system executes a task switch via a Task State Segment (TSS), and the TSS is wholly or partially located within a clean page (A and D bits clear) and the GDT entry for the new TSS is either misaligned across a cache line boundary or is in a clean page, the accessed and dirty bits for an incorrect page table/directory entry may be set.**Implication:** An operating system that uses hardware task switching (or hardware task management) may encounter this erratum. The effect of the erratum depends on the alignment of the TSS and ranges from no anomalous behavior to unexpected errors.

Workaround: The operating system could align all TSSs to be within page boundaries and set the A and D bits for those pages to avoid this erratum. The operating system may alternately use software task management.

Status: For the stepping affected see the *Summary of Changes* at the beginning of this section.

A75. Unsynchronized Cross-Modifying Code Operations Can Cause Unexpected Instruction Execution Results

Problem: The act of one processor, or system bus master, writing data into a currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called cross-modifying code (XMC). XMC that does not force the second processor to execute a synchronizing instruction prior to execution of the new code is called unsynchronized XMC.

Software using unsynchronized XMC to modify the instruction byte stream of a processor can see unexpected instruction execution from the processor which is executing the modified code.

Implication: In this case, the phrase "unexpected execution behavior" encompasses the generation of most of the exceptions listed in the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide* including a General Protection Fault(GPF). In the event of a GPF the application executing the unsynchronized XMC operation would be terminated by the operating system.

Workaround: In order to avoid this erratum, programmers should use the XMC synchronization algorithm as detailed in the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide*, Section 7.1.3.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A76. Deadlock May Occur Due To Illegal-Instruction/Page-Miss Combination

Problem: Intel's 32-bit Instruction Set Architecture (ISA) utilizes most of the available op-code space; however some byte combinations remain undefined and are considered illegal instructions. Intel processors detect the attempted execution of illegal instructions and signal an exception. This exception is handled by the operating system and/or application software.

Under a complex set of internal and external conditions involving illegal instructions, a deadlock may occur within the processor. The necessary conditions for the deadlock involve:

1. The illegal instruction is executed.
2. Two page table walks occur within a narrow timing window coincident with the illegal instruction.

Implication: The illegal instructions involved in this erratum are unusual and invalid byte combinations that are not useful to application software or operating systems. These combinations are not normally generated in the course of software programming, nor are such sequences known by Intel to be generated in commercially available software and tools. Development tools (compilers, assemblers) do not generate this type of code sequence, and will normally flag such a sequence as an error. If this erratum occurs, the processor deadlock condition will occur and result in a system hang. Code execution cannot continue without a system RESET.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A77. FLUSH# Assertion Following STPCLK# May Prevent CPU Clocks From Stopping

Problem: If FLUSH# is asserted after STPCLK# is asserted, the cache flush operation will not occur until after STPCLK# is de-asserted. Furthermore, the pending flush will prevent the processor from entering the Sleep state, since the flush operation must complete prior to the processor entering the Sleep state.

Implication: Following SLP# assertion, processor power dissipation may be higher than expected. Furthermore, if the source to the processor's input bus clock (BCLK) is removed, normally resulting in a transition to the Deep Sleep state, the processor may shutdown improperly. The ensuing attempt to wake up the processor will result in unpredictable behavior and may cause the system to hang.

Workaround: For systems that use the FLUSH# input signal and Deep Sleep state of the processor, ensure that FLUSH# is not asserted while STPCLK# is asserted.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A78. Floating-Point Exception Condition May be Deferred

Problem: A floating-point instruction that causes a pending floating-point exception (ES=1) is normally signaled by the processor on the next waiting FP/MMX™ technology instruction. In the following set of circumstances, the exception may be delayed or the FSW register may contain a wrong value:

1. The excepting floating-point instruction is followed by an instruction that accesses memory across a page (4-Kbyte) boundary or its access results in the update of a page table dirty/access bit.
2. The memory accessing instruction is immediately followed by a waiting floating-point or MMX technology instruction.
3. The waiting floating-point or MMX technology instruction retires during a one-cycle window that coincides with a sequence of internal events related to instruction cache line eviction.

Implication: The floating-point exception will not be signaled until the next waiting floating-point/MMX technology instruction. Alternatively it may be signaled with the wrong TOS and condition code values. This erratum has not been observed in any commercial software applications.

Workaround: None identified

Status: For the stepping affected see the *Summary of Changes* at the beginning of this section.

A79. Snoop Probe During FLUSH# Could Cause L2 to be Left In Shared State

Problem: During a L2 FLUSH operation using the FLUSH# pin, it is possible that a read request from a bus agent or other processor to a valid line will leave the line in the Shared state (S) instead of the Invalid state (I) as expected after flush operation. Before the FLUSH operation is completed, another snoop request to invalidate the line from another agent or processor could be ignored, again leaving the line in the Shared state.

Implication: Current desktop and mid range server systems have no mechanism to assert the flush pin and hence are not affected by this erratum. A high-end server system that does not suppress snoop traffic before the assertion of the FLUSH# pin may cause a line to be left in an incorrect cache state.

Workaround: Affected systems (those capable of asserting the FLUSH# pin) should prevent snoop activity on the front side bus until invalidation is completed after asserting FLUSH#, or use a WBINVD instruction instead of asserting the FLUSH# pin in order to flush the cache.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A80. Livelock May Occur Due to IFU Line Eviction

Problem: Following the conditions outlined for erratum A72, if the instruction that is currently being executed from the evicted line must be restarted by the IFU, and the IFU receives another partial hit on a previously executed (but not as yet completed) store that is resident in the store buffer, then a livelock may occur.

Implication: If this erratum occurs, the processor will hang in a live lock-situation, and the system will require a reset to continue normal operation

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A81. Selector for the LTR/LLDT Register May Get Corrupted

Problem: The internal selector portion of the respective register (TR, LDTR) may get corrupted, if during a small window of LTR or LLDT system instruction execution, the following sequence of events occurs:

1. Speculative write to a segment register that might follow the LTR or LLDT instruction
2. The read segment descriptor of LTR/LLDT operation spans a page (4 Kbytes) boundary; or causes a page fault

Implication: Incorrect selector for LTR, LLDT instruction could be used after a task switch.

Workaround: Software can insert a serializing instruction between the LTR or LLDT instruction and the segment register write.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A82. INIT Does Not Clear Global Entries in the TLB

Problem: INIT may not flush a TLB entry when:

1. The processor is in protected mode with paging enabled and the page global enable flag is set (PGE bit of CR4 register)
2. G bit for the page table entry is set
3. TLB entry is present in TLB when INIT occurs

Implication: Software may encounter unexpected page fault or incorrect address translation due to a TLB entry erroneously left in TLB after INIT.

Workaround: Write to CR3, CR4 or CR0 registers before writing to memory early in BIOS code to clear all the global entries from TLB.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A83. VM Bit will be Cleared on a Double Fault Handler

Problem: Following a task switch to a Double Fault Handler that was initiated while the processor was in virtual-8086 (VM86) mode, the VM bit will be incorrectly cleared in EFLAGS.

Implication: When the OS recovers from the double fault handler, the processor will no longer be in VM86 mode.

Workaround: None identified

Status: For the steppings affected see the *Summary of Errata* at the beginning of this section.

A84. Memory Aliasing with inconsistent A and D Bits May Cause Processor Deadlock

Problem: In the event that software implements memory aliasing by having two Page Directory Entries (PDEs) point to a common Page Table Entry (PTE) and the accessed and dirty bits for the two PDEs are allowed to become inconsistent the processor may become deadlocked.

Implication: This erratum has not been observed with commercially available software.

Workaround: Software that needs to implement memory aliasing in this way should manage the consistency of the accessed and dirty bits.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A85. Use of Memory Aliasing with Inconsistent Memory Type May Cause System Hang

Problem: Software that implements memory aliasing by having more than one linear address mapped to the same physical page with different cache types may cause the system to hang. This would occur if one of the addresses is non-cacheable used in code segment and the other a cacheable address. If the cacheable address finds its way in instruction cache, and non-cacheable address is fetched in IFU, the processor may invalidate the non-cacheable address from the fetch unit. Any micro-architectural event that causes instruction restart will expect this instruction to still be in fetch unit and lack of it will cause system hang.

Implication: This erratum has not been observed with commercially available software.

Workaround: Although it is possible to have a single physical page mapped by two different linear addresses with different memory types, Intel has strongly discouraged this practice as it may lead to undefined results. Software that needs to implement memory aliasing should manage the memory type consistency.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A86. Processor may Report Invalid TSS Fault Instead of Double Fault During Mode C Paging

Problem: When an operating system executes a task switch via a Task State Segment (TSS) the CR3 register is always updated from the new task TSS. In the mode C paging, once the CR3 is changed the processor will attempt to load the PDPTRs. If the CR3 from the target task TSS or task switch handler TSS is not valid then the new PDPTR will not be loaded. This will lead to the reporting of invalid TSS fault instead of the expected Double fault.

Implication: Operating systems that access an invalid TSS may get invalid TSS fault instead of a double fault.

Workaround: Software needs to ensure any accessed TSS is valid.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A87. Machine Check Exception May Occur When Interleaving Code Between Different Memory Types

Problem: A small window of opportunity exists where code fetches interleaved between different memory types may cause a machine check exception. A complex set of micro-architectural boundary conditions is required to expose this window.

Implication: Interleaved instruction fetches between different memory types may result in a machine check exception. The system may hang if machine check exceptions are disabled. Intel has not observed the occurrence of this erratum while running commercially available applications or operating systems.

Workaround: Software can avoid this erratum by placing a serializing instruction between code fetches between different memory types.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A1AP. APIC Access to Cacheable Memory Causes SHUTDOWN

Problem: APIC operations that access memory with any type other than uncacheable (UC) are illegal. If an APIC operation to a memory type other than UC occurs and Machine Check Exceptions (MCEs) are disabled, the processor will enter shutdown after such an access. If MCEs are enabled, an MCE will occur. However, in this circumstance, a second MCE will be signaled. The second MCE signal will cause the Pentium II processor to enter shutdown.

Implication: Recovery from a PIC access to cacheable memory will not be successful. Software that accesses only UC type memory during APIC operations will not encounter this erratum.

Workaround: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A2AP. 2-Way MP Systems May Hang Due to Catastrophic Errors During BSP Determination

Problem: In 2-way MP systems, a catastrophic error during the bootstrap processor (BSP) determination process should cause the assertion of IERR#. If the catastrophic error is due to the APIC data bus being stuck at electrical zero, then the system hangs without asserting IERR#.

Implication: 2-way MP systems may hang during boot due to a catastrophic error. This erratum has not been observed to date in a typical commercial system, but was found during focused system testing using a grounded APIC data bus.

Workaround: None identified

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

A3AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

Problem: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the Pentium II processor despite the attempt to mask it via the LVT.

Implication: Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.

Workaround: Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.

Status: For the steppings affected see the *Summary of Changes* at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the following documents:

- *P6 Family of Processors Hardware Developer's Manual*
- *Pentium® II Processor Developer's Manual*
- *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet
- *Pentium® II Processor at 350 MHz, 400 MHz, and 450 MHz* datasheet
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2, and 3*

All Documentation Changes will be incorporated into a future version of the appropriate Pentium II processor documentation.

A1. 400 MHz S.E.C.C.2 PLGA Part Spec Addition

The shaded row in the following table should be added to Table 28 in the *Pentium® II Processor at 350 MHz, 400 MHz and 450 MHz* datasheet:

Table 28. Thermal Specifications for S.E.C.C.2 Packaged Processors[1]

Processor Core Freq (MHz)	L2 Cache Size (Kbytes)	Processor Power[2] (W)	MIN PLGA T_{CASE} (°C)	MAX PLGA T_{CASE} (°C)	L2 Cache Min T_{CASE} (°C)	L2 Cache Max T_{CASE} (°C)	Min T_{COVER} (°C)	Min T_{COVER} (°C)
400	512	24.3	5	80	5	105	5	75

A2. STPCLK# Pin Definition

The *P6 Family of Processors Hardware Developer's Manual, the Pentium® II Processor Developer's Manual* the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz and 333 MHz* datasheet, and the *Pentium® II Processor at 350 MHz, 400 MHz, and 450 MHz* datasheet all have incorrect definitions of the STPCLK# pin in their alphabetical signal listing. These documents incorrectly state:

> The processor continues to snoop bus transactions and *service interrupts* while in Stop Grant state. When STPCLK# is deasserted, the processor restarts its internal clock to all units and resumes execution.

They should state:

> The processor continues to snoop bus transactions and *may latch interrupts* while in Stop Grant state. When STPCLK# is deasserted, the processor restarts its internal clock to all units, resumes execution, *and services any pending interrupts.*

A3. Invalidating Caches and TLBs

Section 2.6.4 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* incorrectly states:

> The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and TLBs and sends a signal to the external caches indicating that they should be invalidated also.

It should state:

> The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and sends a signal to the external caches indicating that they should be invalidated also.

A4. Handling of Self-Modifying and Cross-Modifying Code

Section 7.1.3 paragraph 1 of the *Intel Architecture Software Developer's Manual Vol 3: System Programming* incorrectly states:

> The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. Intel Architecture processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified. As processor architectures become more complex and start to speculatively execute code ahead of the retirement point (as in the P6 family processors), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future Intel Architectures one of the following two coding options **should** be chosen.

It should state:

> The act of a processor writing data into a currently executing code segment with the intent of executing that data as code is called **self-modifying code**. Intel Architecture processors exhibit model-specific behavior when executing self-modified code, depending upon how far ahead of the current execution pointer the code has been modified. As processor architectures become more complex and start to speculatively execute code ahead of the retirement point (as in the P6 family processors), the rules regarding which code should execute, pre- or post-modification, become blurred. To write self-modifying code and ensure that it is compliant with current and future Intel Architectures one of the following two coding options **must** be chosen.

Section 7.1.3 paragraph 6 of the *Intel Architecture Software Developer's Manual Vol 3: System Programming* incorrectly states:

> The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code**. As with self-modifying code, Intel Architecture processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified. To write cross-modifying code and insure that it is compliant with current and future Intel Architectures, the following processor synchronization algorithm **should** be implemented.

It should state:

> The act of one processor writing data into the currently executing code segment of a second processor with the intent of having the second processor execute that data as code is called **cross-modifying code.** As with self-modifying code, Intel Architecture processors exhibit model-specific behavior when executing cross-modifying code, depending upon how far ahead of the executing processors current execution pointer the code has been modified. To write cross-modifying code and insure that it is compliant with current and future Intel Architectures, the following processor synchronization algorithm **must** be implemented.

A5. Machine Check Architecture Initialization of MCi_STATUS Registers

Section 12.5, the last paragraph of the *Intel Architecture Software Developer's Manual Vol. 3: System Programming Guide* incorrectly states:

> The processor can write valid information (such as an ECC error) into the MCi_STATUS registers while it is being powered up. As part of the initialization of the MCE exception handler, software might examine all the MCi_STATUS registers and log the contents of them, then rewrite them all to zeroes. This procedure is not included in the initialization pseudocode in Example 12-1.

It should state:

> The processor can write valid information (such as an ECC error) into the MCi_STATUS registers while it is being powered up. As part of the initialization of the MCE exception handler, software might examine all the MCi_STATUS registers and log the contents of them, then rewrite them all to zeroes. Following power cycling, the MCi_STATUS registers are not guaranteed to have valid data until after the registers are initially cleared to all zeroes by software. This procedure is not included in the initialization pseudocode in Example 12-1.

A6. LOCK# Signal Prefix Operands

Page 3-273, the first sentence of the third paragraph of the *Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference* states:

> The LOCK prefix can be prepended only to the following instructions and to those forms of the instructions that use a memory operand: ADD, ADC, AND, …

It should state:

> The LOCK prefix can be prepended only to the following instructions and to those forms of the instructions that use a memory operand as the destination operand only: ADD, ADC, AND, ...
>
> If the LOCK prefix is used with the memory operand as the source operand then an Invalid Opcode (Undefined Opcode), #UD, can occur.

A7. SMRAM State Save Map Contains Documentation Errors

In the *Intel Architecture Software Developer's Manual Volume 3: System Programming Guide*, revision 2 Chapter 12, "System Management Mode (SMM)," Table 12-1 incorrectly documents the SMBASE+Offset for LDT Base on the P6 family of processors.

> The storage locations for these parameters are model specific (i.e., they may differ between the Pentium® processor, the Pentium® Pro processor, and other P6 family proliferations). **These entries in the tables above will be changed to Reserved. Hardware and software may not rely on the contents of these Reserved regions.**

A8. Memory Aliasing with Different Memory Types

The 4th paragraph of section 9.13.5 of Intel Architecture Software Developer's Manual Vol. 3: Programming the PAT states:

> The PAT allows any memory type to be specified in the page table, and therefore it is possible to have a single physical page mapped to two different linear addresses with different memory types. This practice is strongly discouraged by Intel and should be avoided as it may lead to undefined results.

It should change to:

> The PAT allows any memory type to be specified in the page table, and therefore it is possible to have a single physical page mapped to two different linear addresses with different memory types. Intel does not support this practice as it may lead to undefined operations including processor hang.

A9. System Management Interrupt (SMI) During Start-up IPI Clarification

The note on section 12.2 of Intel Architecture Software Developer's Manual Vol. 3: System Management Interrupt (SMI) states:

> In the P6 family of processors, when a processor that is designated as the application processor during an MP initialization protocol is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked.

It should state:

> In the P6 family of processors, when a processor that is designated as the application processor during an MP initialization protocol is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked. However, if SMI is received while an application processor is in the wait for SIPI mode, it will be pended. The processor will respond on receipt of a SIPI by immediately servicing the pended SMI and will go into SMM before executing from the SIPI vector.

A10. Runbist will Not Function When STPCLK# Driven Low

Paragraph 5 of Section 6.3 in the *P6 Family of Processors Hardware Developer's Manual* currently states:

> Note that RUNBIST will not function when the processor core clock has been stopped. All other 1149.1-defined instructions operate independently of the processor core clock.

It should state:

> Note that RUNBIST will not function when the processor STPCLK# input has been driven low. All other 1149.1-defined instructions will correctly operate regardless of the STPCLK# signal state.

A11. Memory Aliasing with Inconsistent A and D Bits may Cause Processor Deadlock

Add the following note to Chapter 3 and Chapter 9.13.5 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Revision 2:

> The processor allows memory aliasing by having two Page Directory Entries (PDEs) point to a common Page Table Entry (PTE). Software that needs to implement memory aliasing in this way should manage the consistency of the Accessed and Dirty bits. Allowing the Accessed and Dirty bits for the two PDEs to become inconsistent may lead to a processor deadlock.

A12. An Interrupt Could Occur While TSS is Marked Busy

Paragraph 5 of section 6.1.3 in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, currently states:

> For all Intel Architecture processors, tasks are not recursive. A task cannot call or jump to itself.

It should state:

> For all Intel Architecture processors, tasks are not recursive. A task cannot call or jump to itself. Because Intel Architecture tasks are not re-entrant, a task used as an interrupt handler must have interrupts disabled between the time it completes the interrupt and the time it exits with IRET. Otherwise, another interrupt could occur while the interrupt task's TSS is still marked busy, causing a #GP fault.

A13. NMI Unmasked Early When Processor is Running in V86 Mode

Section 5.5.1 in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, currently states:

> While an NMI interrupt handler is executing, the processor disables additional calls to the NMI handler until the next IRET instruction is executed. This blocking of subsequent NMIs prevents stacking up calls to NMI handler. It is recommended that the NMI interrupt handler be accessed through an interrupt gate to disable maskable hardware interrupts (refer to section 5.6.1, "Masking Maskable Hardware Interrupts").

It should state:

> While an NMI interrupt handler is executing, the processor disables additional calls to the NMI handler until the next IRET instruction is executed. This blocking of subsequent NMIs prevents stacking up calls to NMI handler. It is recommended that the NMI interrupt handler be accessed through an interrupt gate to disable maskable hardware interrupts (refer to section 5.6.1, "Masking Maskable Hardware Interrupts"). If the NMI handler is a virtual-8086 task with IOPL less than 3, the IRET from this handler triggers a general-protection exception (#GP) (section 16.2.7). In this case, NMI is unmasked before the #GP handler is invoked.

A14. P6 Reads Two Bytes for POP SEG Instruction

Paragraph 1 and 2 of section 18.24.1 in *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, currently states:

> When pushing a segment selector onto the stack, the Intel486(TM) processor writes 2 bytes onto 4-byte stacks and decrements ESP by 4. The P6 family and Pentium® processors write 4 bytes, with the upper 2 bytes being zeros.
>
> When popping a segment selector from the stack, the Intel486(TM) processor reads only 2 bytes. The P6 family and Pentium® processors read 4 bytes and discard the upper 2 bytes. This operation may have an effect if the ESP is close to the stack-segment limit. On the P6 family and Pentium® processors, stack location at ESP plus 4 may be above the stack limit, in which case a stack fault exception (#SS) will be generated. On the Intel486(TM) processor, stack location at ESP plus 2 may be less than the stack limit and no exception is generated.

It should state:

> When pushing a segment selector, Intel486(TM) and P6 family processors decrement ESP by the operand size and then write two bytes. If the operand size is 32-bits, the upper two bytes of the write are unmodified. The Intel Pentium® processor decrements ESP by the operand size and determines the size of the write by the operand size. If the operand size is 32-bits, the upper two bytes of the write are zero.
>
> When popping a segment selector, Intel486(TM) and P6 family processors read two bytes and increment ESP by the operand size of the instruction. The Intel Pentium® processor determines the size of the read by the operand size and increments ESP by the operand size.
>
> It is possible to align a 32-bit selector push or pop such that the operation will generate an exception on the Intel Pentium® processor and not on the Intel486(TM) and P6 family processors. This could occur if the third and/or fourth byte of the operation lies beyond the limit of the segment or if the third and/or fourth byte of the operation is located on a not-present or inaccessible page.

A15. APIC Register Offsets are Aligned on 128-bit Boundaries

Paragraph 3 of section 7.5.7 in *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, currently states:

> Within the 4KB APIC register area, the register address allocation scheme is shown in Table 7-1. Register offsets are aligned on 128-bit boundaries. All registers must be accessed using 32-bit loads and stores. Wider registers (64-bit or 256-bit) are defined and accessed as independent multiple 32-bit registers. If a LOCK prefix is used with a MOV instruction that accesses the APIC address space, the prefix is ignored; that is a locking operation does not take place.

It should say:

> Within the 4KB APIC register area, the register address allocation scheme is shown in Table 7-1. Register offsets are aligned on 128-bit boundaries. All registers must be accessed using loads and stores that are aligned on 128-bit boundaries and manipulate the registers as 32-bit quantities. Wider registers (64-bit or 256-bit) are defined and accessed as independent multiple 32-bit registers. If a LOCK prefix is used with a MOV instruction that accesses the APIC address space, the prefix is ignored; that is a locking operation does not take place.

A15. APIC Register Offsets are Aligned on 128-bit Boundaries

Paragraph 3 of section 7.5.7 in *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, currently states:

> Within the 4KB APIC register area, the register address allocation scheme is shown in Table 7-1. Register offsets are aligned on 128-bit boundaries. All registers must be accessed using 32-bit loads and stores. Wider registers (64-bit or 256-bit) are defined and accessed as independent multiple 32-bit registers. If a LOCK prefix is used with a MOV instruction that accesses the APIC address space, the prefix is ignored; that is a locking operation does not take place.

It should say:

> Within the 4KB APIC register area, the register address allocation scheme is shown in Table 7-1. Register offsets are aligned on 128-bit boundaries. All registers must be accessed using loads and stores that are aligned on 128-bit boundaries and manipulate the registers as 32-bit quantities. Wider registers (64-bit or 256-bit) are defined and accessed as independent multiple 32-bit registers. If a LOCK prefix is used with a MOV instruction that accesses the APIC address space, the prefix is ignored; that is a locking operation does not take place.

A16. Single Stepping of Instructions Breaks out of HLT State

Paragraph 1 of section 2.6.5 in *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, currently states:

> The HLT (halt processor) instruction stops the processor until an enabled interrupt (such as NMI or SMI, which are normally enabled), the BINIT# signal, the INIT# signal, or the RESET# signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered. Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked. (Note that the BINIT# pin was introduced with the Pentium® Pro processor)

It should say:

> The HLT (halt processor) instruction stops the processor until an enabled interrupt (such as NMI, or SMI which are normally enabled), a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered. Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked. (Note that the BINIT# pin was introduced with the Pentium® Pro processor)

A17. Additional Signal Resumes Execution While in a HALT State

Paragraph 1 of the Description section of page 3-291 in *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual*, currently states:

> This instruction stops instruction execution and places the processor in a HALT state. An enabled interrupt, NMI, or a reset will resume execution. If an interrupt (including NMI) is used to resume execution after a HLT instruction, the saved instruction pointer (CS:EIP) points to the instruction following the HLT instruction.

It should say:

> This instruction stops instruction execution and places the processor in a HALT state. An enabled interrupt (including NMI and SMI), a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal will resume execution. If an interrupt (including NMI) is used to resume execution after a HLT instruction, the saved instruction pointer (CS:EIP) points to the instruction following the HLT instruction.

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the following documents:

- *P6 Family of Processors Hardware Developer's Manual*
- *Pentium® II Processor Developer's Manual*
- *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet
- *Pentium® II Processor at 350 MHz, 400 MHz, and 450 MHz* datasheet
- *Intel Architecture Software Developer's Manual*, *Volumes 1, 2, and 3*

All Specification Clarifications will be incorporated into a future version of the appropriate Pentium II processor documentation.

A1. PWRGOOD Inactive Pulse Width

Footnote 8 of Table 13 in the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet, should read as follows:

8. When driven inactive or after V_{CCCORE}, V_{CCL2}, and BCLK become stable. PWRGOOD must remain below V_{IL},max from Table 7 until all the voltage planes meet the voltage tolerance specifications in Table 6 and BCLK has met the BCLK AC specifications in Table 10 for at least 10 clock cycles. PWRGOOD must rise glitch-free and monotonically to 2.5 V.

A2. MTRR Initialization Clarification

The following sentence should be added to the end of the first paragraph of Section 9.12.5 of *the Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*: "The MTRRs must be disabled prior to initialization or modification."

A3. Floating-Point Opcode Clarification

Section 3.2 of the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference* provides detailed descriptions of each Intel Architecture instruction. For some instructions, the clarification phrase below needs to be either added to their existing "Comments" section or a "Comments" section needs to be created with the clarification phrase. The phrase is as follows:

> The (**Instruction** shown in the center column of the table below) instruction is actually a combination of two instructions – the wait instruction followed by (**Instruction** shown in the table). If the (**Instruction** shown in the table) instruction should fault in some way (e.g., page fault), the value of EIP that is passed to the fault handler will be equal to the EIP of the first instruction plus one (i.e., the EIP of the second of the pair of instructions). The FWAIT portion of the combined instruction will have completed execution and will typically not be, nor need to be, re-executed after the fault handler is completed.

The following table lists the affected instructions and the location of the clarification phrase:

Instruction Set Reference Section	Opcode	Instruction	Clarification	Page
FCLEX/FNCLEX-Clear Exceptions	9B DB E2	FCLEX	Add "Comments" section with clarification phrase	3-177
FINIT/FNINIT-Initialize Floating-Point Unit	9B DB E3	FINIT	Add clarification phrase to existing "Comments" section	3-204
FSAVE/FNSAVE-Store FPU State	9B DD /6	FSAVE m94/108byte	Add clarification phrase to existing "Comments" section	3-237
FSTCW/FNSTCW-Store Control Word	9B D9 /7	FSTCW m2byte	Add "Comments" section with clarification phrase	3-250
FSTENV/FNSTENV-Store FPU Environment	9B D9 /6	FSTENV m14/28byte	Add "Comments" section with clarification phrase	3-253
FSTSW/FNSTSW-Store Status Word	9B DD /7	FSTSW m2byte	Add "Comments" section with clarification phrase	3-256
	9B DF E0	FSTSW AX		

A4. Non-AGTL+ Output Low Current Clarification

In Table 8 of the *Intel® Pentium® II Processor at 233MHz, 266MHz, 300MHz, and 333MHz* datasheet, the note in **bold** should be added:

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	-0.3	0.7	V	

Symbol	Parameter	Min	Max	Unit	Notes
V_{IH}	Input High Voltage	1.7	2.625	V	2.5 V +5% maximum
V_{OL}	Output Low Voltage		0.4	V	1
V_{OH}	Output High Voltage	N/A	2.5	V	All outputs are open-drain to 2.5 V +5%
I_{OL}	Output Low Current	14		mA	**4**
I_{LI}	Input Leakage Current		±100	µA	2
I_{LO}	Output Leakage Current		±15	µA	3

Notes:

1 Parameter measured at 14 mA (for use with TTL inputs).

2. ($0 \le V_{IN} \le 2.5$ V +5%).

3. ($0 \le V_{OUT} \le 2.5$ V +5%).

4. **Specified as the minimum amount of current that the output buffer must be able to sink. However, VOL_MAX cannot be guaranteed if this specification is exceeded.**

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the following documents:

- *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet
- *Pentium® II Processor at 350 MHz, 400 MHz, and 450 MHz* datasheet

All Specification Changes will be incorporated into a future version of the appropriate Pentium II processor documentation.

A1. FRCERR Pin Removed From Specification

The Pentium II processor will not use the FRCERR pin. All references to these pins will be removed from the specification. These references currently appear in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Appendix B. These references also appear in the *Pentium® II Processor Developer's Manual*, Sections 3.2.7, 5.1.14, 7.5, 7.12 and A.1.23. Finally, these references appear in the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet, Sections 2.8, 2.13, and A.1.23.

A2. Non-GTL+ Output Leakage Current Change

The CMOS, TAP, Clock and APIC Signal Group Output Leakage Current specification (I_{LO} in Table 8 of the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet, or the *Pentium® II Processor at 350 MHz, 400 MHz, and 450 MHz* datasheet) should be changed from ±15 µA to ±30 µA.

A3. 350 MHz, 400 MHz and 450 MHz T_{junc} Offset Temperature Spec Addition

The shaded column in Table 28 below and note 3 are additions to the *Pentium® II Processor at 350 MHz, 400 MHz and 450 MHz* datasheet:

Table 28. Thermal Specifications for S.E.C.C.2 Packaged Processors

Processor Core Frequency (MHz)	L2 Cache Size (Kbytes)	Processor Power[2] (W)	Min PLGA T_{CASE} (C)	Max PLGA T_{CASE} (C)	Min OLGA T_{JUNC} (°C)	Max OLGA T_{JUNC} (°C)	T_{JUNC} Offset[3] (°C)	L2 Cache Min T_{CASE} (°C)	L2 Cache Max T_{CASE} (°C)	Min T_{COVER} (°C)	Max T_{COVER} (°C)
450	512	27.1	N/A	N/A	5	90	4.8	5	105	5	75
400	512	24.3	N/A	N/A	5	90	4.8	5	105	5	75
350	512	21.5	5.0	80.0	N/A	N/A	4.8	5	105	5	75

NOTES:

1. These values are specified at nominal V_{CCCORE} for the processor core and nominal V_{CCL2} for the L2 cache.
2. Processor power includes the power dissipated by the processor core, the L2 cache, and the AGTL + bus termination. The maximum power for each of these components does not occur simultaneously.

3. $T_{JUNCTIONOFFSET}$ is the worst-case difference between the thermal reading from the on-die thermal diode and the hottest location on the processor's core.

A4. RESET# Pin Definition

The *P6 Family of Processors Hardware Developer's Manual, the Pentium® II Processor Developer's Manual*, the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz and 333 MHz* datasheet, and the *Pentium® II Processor at 350 MHz, 400 MHz, and 450 MHz* datasheet all have incorrect definitions of the RESET# pin in their alphabetical signal listing. These documents incorrectly state:

> RESET# must remain active for one microsecond for a 'warm' Reset; for a Power-on Reset, RESET# must stay active for at least one millisecond after V_{CCCORE} and CLK have reached their proper specifications.

They should state:

> For a Power-on or 'warm' reset, RESET# must stay active for at least one millisecond after V_{CCCORE} and CLK have reached their proper specifications.